ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA INFORMÁTICA

ANÁLISIS DE MUTACIONES PARA COMPOSIONES DE SERVICIOS WEB EN WS-BPEL 2.0

- Koura -

Lorena Gutiérrez Madroñal

Cádiz 2009

ESCUELA SUPERIOR DE INGENIERÍA
INGENIERÍA INFORMÁTICA

ANÁLISIS DE MUTACIONES PARA COMPOSIONES DE SERVICIOS WEB EN WS-BPEL 2.0
- Koura -

DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Director del Proyecto: Francisco Palomo Lozano
Autor del Proyecto: Lorena Gutiérrez Madroñal

Cádiz 2009
Fdo: Lorena Gutiérrez Madroñal.

Agradecimientos

Al grupo de investigación de la Universidad de Cádiz SPI&FM que me ha brindado la oportunidad de aprender junto a ellos nuevos conocimientos, aprender nuevas formas de trabajo y una nueva forma de hacer amigos.

A todos aquellos que se preocuparon por mi no solo en mi vida profesional, gracias a ellos que me dieron un empujón en esta etapa tan difícil de mi otra vida.

A mi familia y amigos, que de algún modo u otro estuvieron ahí, unos más y otros menos, en los pasos y tropiezos que he estado dando a lo largo de estos años.

A todas las personillas especiales en mi vida, especialmente a ellos (valga la redundancia), gracias.

Índice general

Índice de figuras

Capítulo 1 Introducción

1.1. Motivación

La realización de este proyecto ha sido llevada a cabo gracias a la beca de colaboración con el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz. Dentro de este departamento se está investigando sobre el lenguaje WS-BPEL y su rango de aplicación, más concretamente en los servicios web.

El grupo de investigación SPI&FM se centró en el siguiente estudio para enfocar la investigación que están llevando a cabo:
El World Wide Web Consortium (W3C) define un servico Web (WS) como una aplicación software identificada por un URI cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML. Los WS permiten la interoperación de sistemas distribuidos heterogéneos con independencia de las plataformas hardware y software empleadas; esto hace que tengan una gran definición de estándares relacionados con ellos. Los principales estándares relacionados con los WX son SOAP, protocolo que permite especificar la estructura de los mensajes que intercambian los WS; WDSL, lenguaje que permite especificar los WS; UDDI, que permite publicar y descubrir WS, y WS-BPEL, lenguaje que permite crear procesos de negocio mediante la composición de WS preexistentes y ofrecernos a su vez como WS [1].

La importancia económica que están alcanzando las composiciones de servicios en WS-BPEL obliga a que se preste especial atención a la prueba de este tipo de software. hasta la fecha se han publicado trabajos relacionados con diversos aspectos de la prueba de composiciones de WS en WS-BPEL, la mayor parte de ellos relacionados con la generación de casos de prueba. La mayoría de estos trabajos no estudian la calidad de los casos de prueba generados [1].

En el grupo de investigación SPI&FM se cree que es interesante disponer de un sistema que les permita medir la calidad de distintos conjuntos de casos de prueba. Una técnica apropiada para realizar esta labor es el análisis de mutaciones (mutation analysis). Esta técnica utiliza una serie de operadores de mutación para generar un conjunto de mutantes a partir del programa a probar. Aunque se han publicado diversos trabajos sobre la definición de operadores de mutación para distintos lenguajes, que comentaremos posteriormente, no se han encontrado ninguno que defina los operadores de mutación para el lenguaje WS-BPEL [1].

El objetivo principal del trabajo del grupo de investigación SPI&FM es definir un conjunto de operadores de mutación para el lenguaje WS-BPEL 2.0 que permita mode-

lar los fallos comunes que pueden cometer los programadores a la hora de escribir un programa en este lenguaje. Durante el proceso de definición de los operadores se prestó especial atención en la eliminación de aquellos que, de manera obvia, podrían provocar la generación de mutantes equivalentes [1].

Como ya hemos visto, el análisis de mutaciones es una técnica adecuada para medir la calidad de los casos de prueba y ha sido ampliamente aplicada a programas escritos en diversos lenguajes. Ya que para poder utilizarla es necesario disponer de un conjunto apropiado de operadores de mutación específicos del lenguaje que determinen los cambios que se van a introducir en el programa a probar, necesitamos poder controlar dichos operadores y manejarlos según el estudio a realizar.

Para facilitar el análisis de mutaciones y obtener de una forma más sencilla la medida de calidad de los casos de prueba, vamos a elaborar una aplicación que nos va a permitir obtener los resultados de una forma más clara, ya que su manejo va a ser más sencillo al que se estaba realizando anteriormente (escribiendo directamente en consola el estudio deseado). También se añaden nuevas funcionalidades a ésta aplicación que nos abrirán nuevos campos para el estudio de las mutaciones.

Ésta aplicación se aplicará a uno de los proyectos de investigación del grupo SPI&FM, el proyecto *GAmera*, del que hablaremos posteriormente en la sección dedicada al mismo.

1.2. Objetivos

Nuestro principal objetivo es el estudio de la herramienta desarrollada por el grupo de investigación SPI&FM para la generación automática de mutantes para el lenguaje WS-BPEL. Y después de esto el desarrollo de una interfaz gráfica para la herramienta de generación automática de mutantes para el lenguaje WS-BPEL, para facilitar el análisis de mutaciones y el control de las iteraciones, los ficheros y los operadores a emplear.

Otro de nuestros objetivos es el estudio de la técnica de análisis de mutaciones del entorno desarrollado para la herramienta μJava. Así que teniendo como referencia el interfaz de la aplicación μJava, vimos que se dividía en tres secciones; la primera de ellas era para la introducción de datos, la segunda para visualizar los resultados obtenidos y la tercera para obtener datos estadísticos de los resultados del análisis.

La primera sección consiste en un listado de ficheros (Java en el caso de la aplicación μJava), en la cual es el usuario el que selecciona los ficheros que contienen los datos y los operadores con los que quiere realizar el análisis. Nosotros tendríamos que hacer lo mismo en nuestra aplicación, pero con los ficheros de WS-BPEL y los operadores que se utilizan en el Departamento de Lenguaje y Sistemas Informáticos para el estudio.

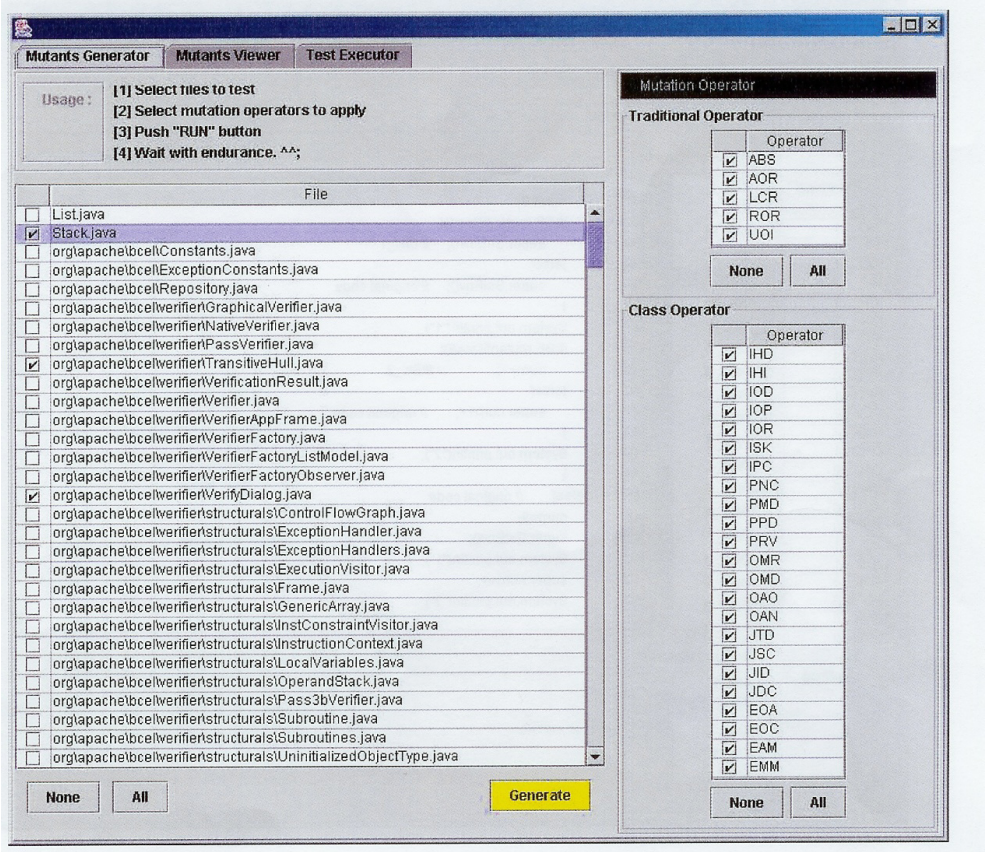

Figura 1.1: Captura μJava, primera sección

En la segunda sección se separan los resultados en dos secciones, diferenciándolos según los operadores empleados: las clases de mutantes y los mutantes tradicionales. Según los operadores que empleemos, dividiremos los resultados para que se obtengan de una forma más clara.

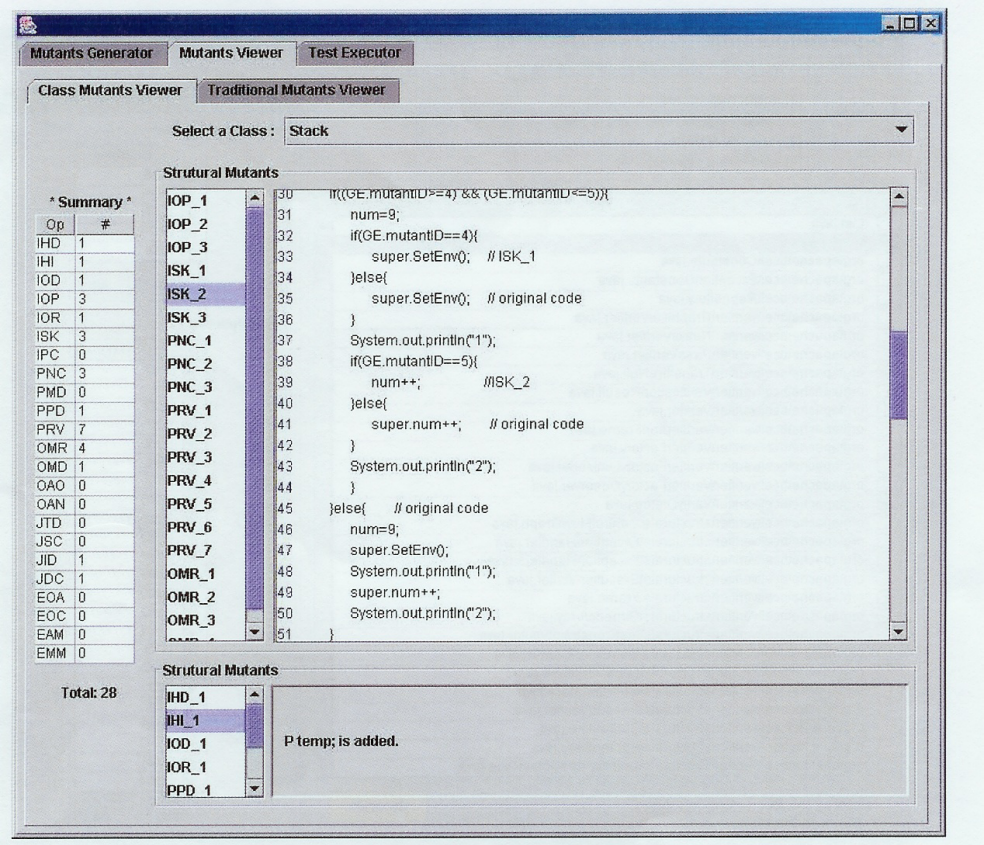

Figura 1.2: Captura μJava, segunda sección

Y finalmente en la tercera sección, los resultados estadísticos los que también aparecen divididos según los operadores empleados. Diseñaremos nuestra interfaz del mismo modo dependiendo de nuestros operadores.

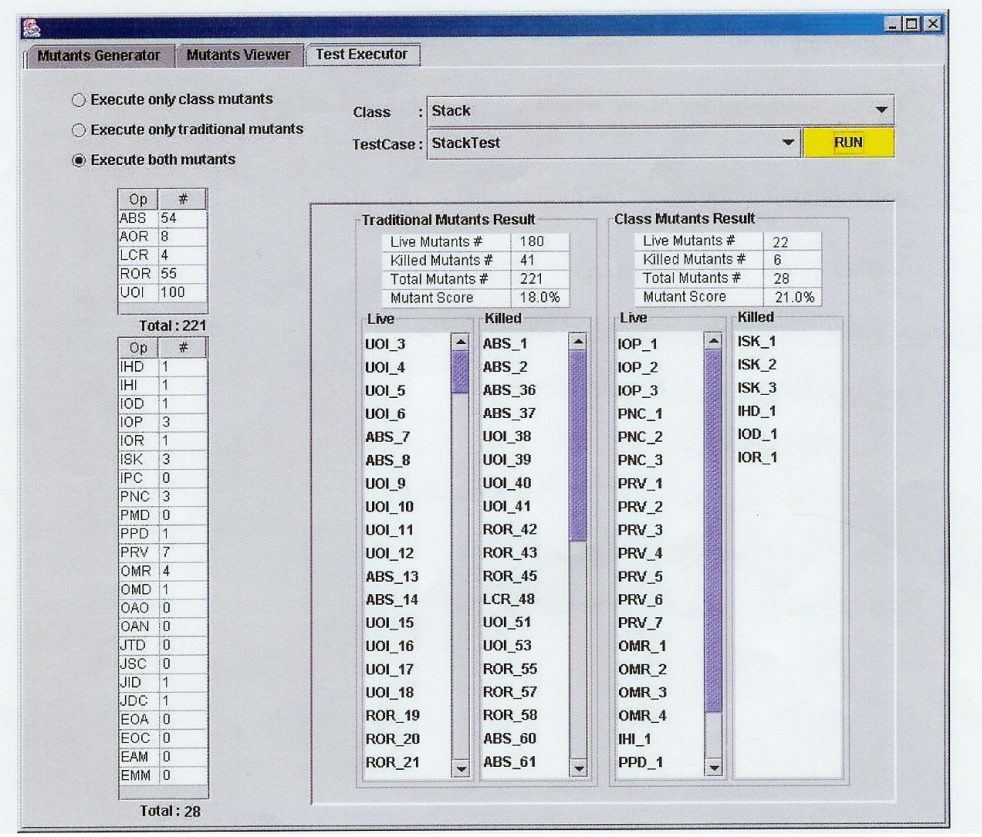

Figura 1.3: Captura μJava, tercera sección

1.3. Definiciones previas

Servicios Web

Permiten el intercambio de información en sistemas distribuidos heterogéneos. Forman una arquitectura que permite describir, publicar, descubrir, solicitar e invocar diferentes servicios empleando la infraestructura de internet.

El lenguaje WS-BPEL, especifica el comportamiento de un proceso de negocio basado en interacciones con Servicios Web.

Algoritmos genéticos

Son técnicas de búsqueda probabilística basadas en la teoría de la evolución y la selección natural.

Supervivencia de los mejores y carácter hereditario de las características. Se favorecen a los mejores y generan nuevos individuos (recombinación y mutación de información).

Análisis de mutaciones

Proceso de medir la calidad de conjuntos de casos de prueba.

Mutantes

Programas que contienen una única diferencia con respecto al programa original. Se generan aplicando al código fuente un conjunto de reglas definidas previamente, los operadores de mutación.

Operadores de mutación

Conjunto de reglas que introducen pequeños cambios sintácticos basados en los errores que suelen cometer habitualmente los programadores, o bien pretenden forzar ciertos criterios de cobertura del código.

Mutantes equivalentes

Mutantes que siempre provocan la misma salida que el programa original, por lo que no va a existir ningún caso de prueba que permita salida que el programa original.

Puntuación de mutación

Cociente entre el número de mutantes muertos y el número de mutantes no equivalentes.

1.4. Antecedentes

El análisis de mutaciones es el proceso de medir la calidad de conjuntos de casos de prueba. Para ello genera un gran número de programas, denominados **mutantes**, que contienen una única diferencia con respecto al programa original. Los mutantes se generan aplicando al código fuente un conjunto de reglas definidas previamente, los **operadores de mutación**, que introducen pequeños cambios sintácticos basados en los errores que suelen cometer habitualmente los programadores, o bien pretenden forzar ciertos criterios de cobertura del código. Estos operadores introducen cambios en el programa a probar manteniendo su validez sintáctica [1]. Así, si un programa tiene la instrucción *compra >5000* y disponemos de operadores de mutación sobre los operadores relacionales, que consisten en cambiar un operador por otro, el mutante resultante podría tener como instrucción, por ejemplo, *compra <5000* [2].

Una vez generados, los mutantes se ejecutan sobre los casos de prueba; si la salida que produce el mutante es diferente de la que produce el programa original sobre un determinado caso de prueba, se dice que el mutante está muerto. En ocasiones, aparecen mutantes que siempre provocan la misma salida que el programa original, por lo que no va a existir ningún caso de prueba que permita matarlos; éstos se denominan **mutantes equivalentes**. Para medir la calidad de un conjunto de casos de prueba debemos eliminar los mutantes equivalentes, ya que ésta se va a calcular mediante la **puntuación de mutación** (mutation score), el cociente entre el número de mutantes muertos y el número de mutantes no equivalentes [1].

$$\mathrm{TM} = \boxed{\frac{D}{M - E}} \tag{1}$$

Donde D es el número de mutantes muertos, M el número total de mutantes y E el número de mutantes equivalentes [2].

Dentro de la prueba de mutaciones, el trabajo presentado por el grupo de investigación SPI&FM, se centra en dos aspectos: por un lado, en el desarrollo de un generador de mutantes y, por otro, en el desarrollo de un generador de casos de prueba [2].

1.4.1. Descripción de la Arquitectura

La arquitectura que se propone presenta una doble funcionalidad: generar mutantes para composiciones de servicios en WS-BPEL, que puede ser extensible a otros lenguajes, y generar casos de prueba. El núcleo de la arquitectura de los dos generadores está basado en algoritmos genéticos. En concreto, se pretenden resolver las siguientes cuestiones [2]:

- Generar automáticamente mutantes
- Generar automáticamente casos de prueba
- Detectar mutantes equivalentes al programa original
- Obtener casos de prueba para matar a mutantes difíciles de matar

En la siguiente figura, se muestra el esquema de la arquitectura. En ella podemos distinguir tres bloques: un bloque de análisis del programa original, un generador de mutantes, y un generador de casos de prueba. La arquitectura tiene como entrada una composición de servicios en WS-BPEL, un conjunto inicial de casos de prueba sobre dicha composición, así como los parámetros de configuración de los algoritmos genéticos [2].

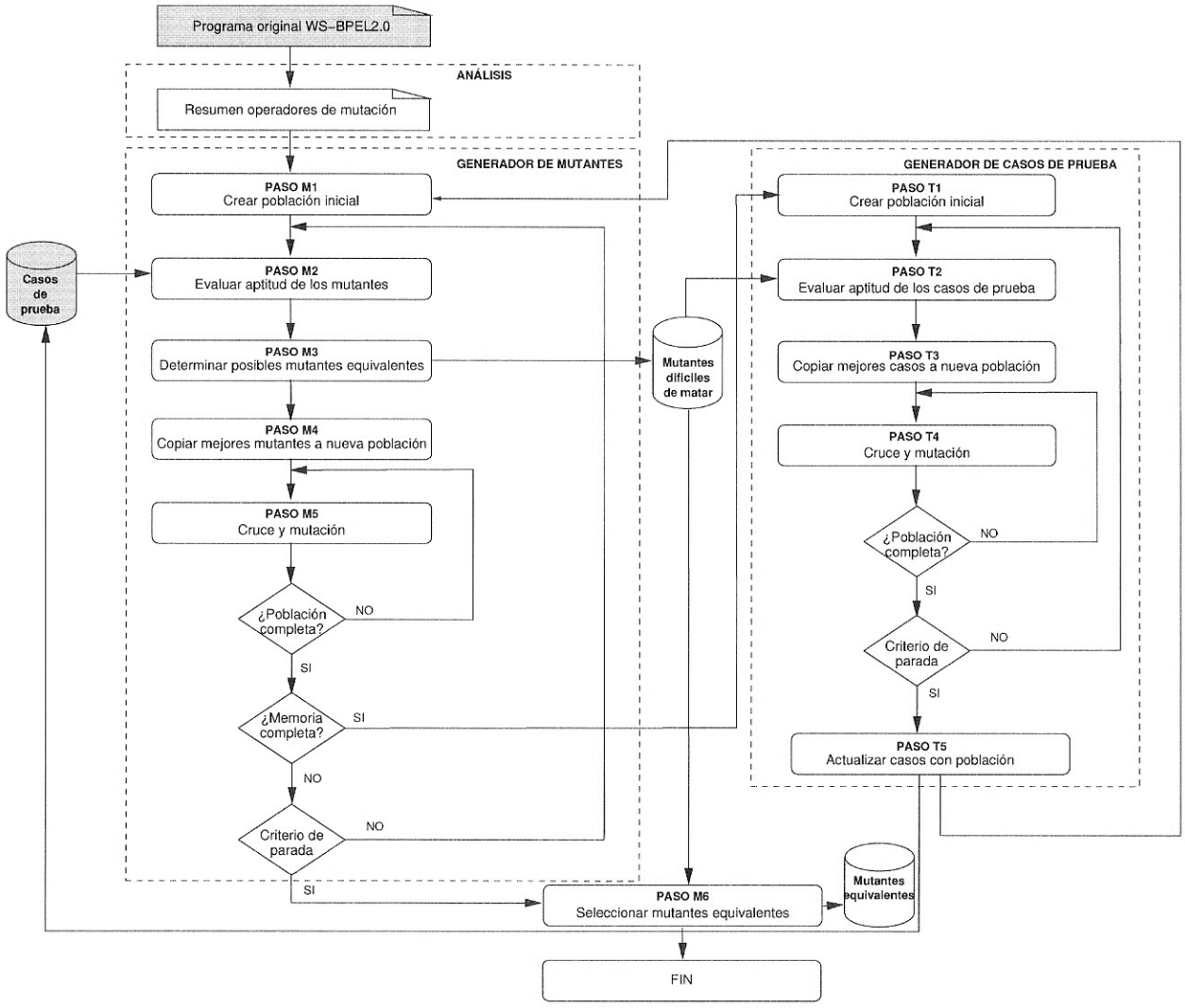

Figura 1.4: Arquitectura para la prueba de mutaciones de composiciones de servicios web

Antes de ejecutar el generador de mutantes, el sistema analizará el programa original WS-BPEL, de manera que se identificarán las instrucciones que pueden sufrir mutación de acuerdo a los operadores de mutación definidos para el lenguaje. A continuación se ejecutará el generador de mutantes, cuyo objetivo es generar mediante un algoritmo genético y de forma automática los mutantes del programa original, localizando los posibles mutantes equivalentes [2].

Por otro lado el generador de los casos de prueba tendrá como misión crear un conjunto de casos de prueba de alta calidad que sea capaz de diferenciar entre mutantes equivalentes y mutantes difíciles de matar generados anteriormente [2].

Una vez ejecutado el generador de casos de prueba encargado de detectar los mutantes equivalentes, el sistema volverá a lanzar el generador de mutantes. Este ciclo se repetirá de forma iterativa hasta que se alcance una puntuación (1) establecida inicialmente [2].

En las siguientes secciones vamos a hablar sobre las diferentes aplicaciones que influyen de una forma u otra en Koura, bien sea para el diseño, algunas funcionalidades, o porque trabajamos sobre ellas.

1.5. Estudio en profundidad de μJava

μJava (muJava) es un sistema de mutación en Java. Genera automáticamente mutantes para cada una de las pruebas; los test "traditional mutation" y "class-level mutation". μJava puede proponer tests de pruebas para clases individuales, o para paquetes de clases. Cada uno de estos tests, son las respuestas a los usuarios, a consecuencia de sus llamadas a los métodos de las clases de prueba, encapsulados en clases separadas. [6]

μJava es el resultado de la colaboración entre dos universidades, "Korea Advanced Institute of Science and Technology (KAIST)" en Corea del Sur y "George Mason University" en los EE.UU.. Este proyecto con colaboradores tales como: Yu Seung Ma, candidato al doctorado en KAIST, el doctor Rae Kwon Yong, profesor de KAIST y el doctor Jeff Offutt, profesor de la universidad George Mason. La mayoría del software de desarrollo fue realizado por YuSeung. [6]

1.5.1. Panorama general

μJava fue creado por Ma, Offutt y Kwon. μJava utiliza dos tipos de operadores de mutación, las clases y los métodos de nivel. Los primeros fueron diseñados para las clases de Java por Ma, Kwon y Offutt, y, a su vez diseñada a partir de una categorización de orientación a objetos por fallas por Offutt, Alexander. μJava crea mutantes orientados a objetos para las clases de Java, de acuerdo a 24 operadores que están especializados en las fallas orientadas a objetos. Los mutantes de métodos a nivel (tradicional), se basan

en el operador selectivo establecido por Offutt et al. Después de crear mutantes, μJava permite al usuario entrar y ejecutar las pruebas, y evalúa la cobertura de la mutación de las pruebas. [6]

En μJava, las pruebas de detección de la clases están codificadas en clases separadas para hacer llamadas a los métodos de las clases de las que se hace la prueba. Los mutantes se crean y ejecutan automáticamente. Los mutantes equivalentes deben ser identificados a mano. [6]

Algo de historia

- 2003. Por primera vez conocido como JMutation (**J**ava **Mutation** System)
- 2004. El nombre fue cambiado a MuJava (**Mu**tation System for **Java**)
- 2005. El software se registra con copyright, todos los derechos reservados
- 2005. La versión 2 aparece con varias correcciones de errores y con operadores mutantes modificados

1.5.2. Generando mutantes con μJava

Después de realizar una serie de pasos, colocando los archivos de origen para las pruebas en ciertos directorios, y de ejecutar una serie de comandos, llegamos a una pantalla similar a la siguiente [6]:

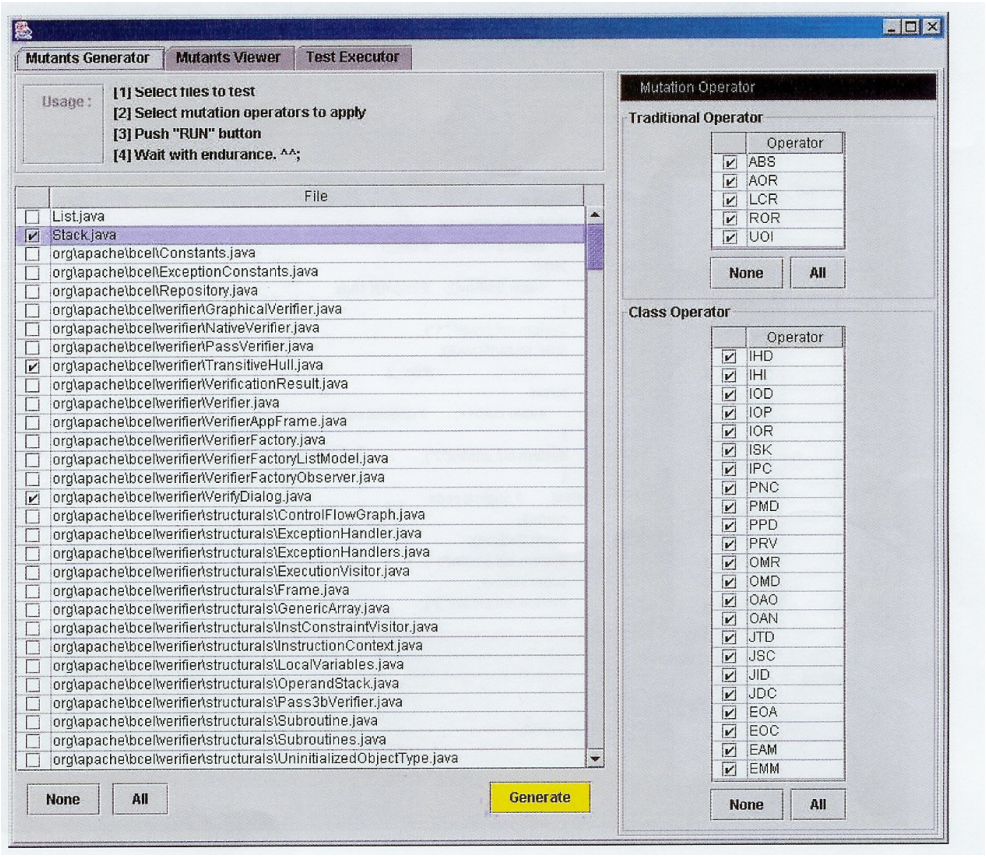

Figura 1.5: Captura μJava, primera sección

Seleccionaremos los archivos que se desea mutar haciendo clic en las casillas de la izquierda. Escogeremos los operadores de mutación que deseamos utilizar, según las cajas de la derecha y finalmente pulsaremos “Generate” [6].

Después de que se generen los mutantes, podemos verlos en las pestañas "Visor de las clases mutantes" y "Visor de los mutantes tradicionales", como se muestra en la siguiente figura [6].

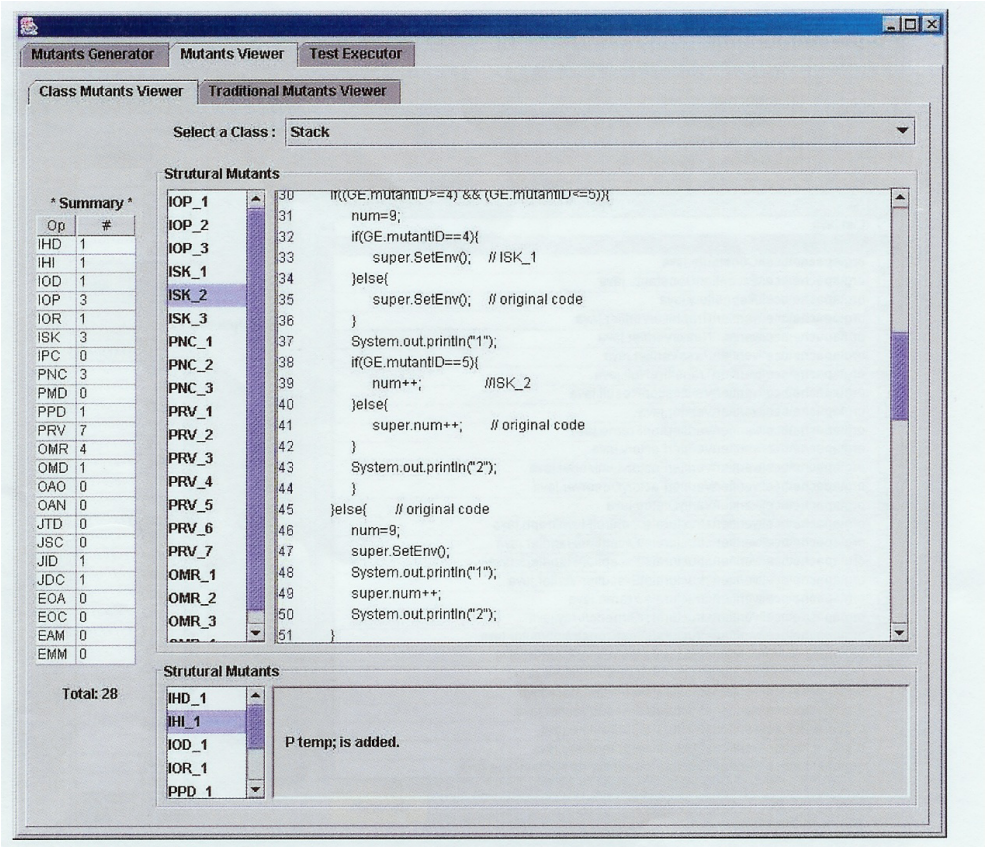

Figura 1.6: Captura μJava, segunda sección

En la siguiente figura, se puede ver que podemos seleccionar qué colección de mutantes va a ser ejecutada y que pruebas vamos a utilizar. Podemos diseñar pruebas para matar a los mutantes que han quedado vivos, y posteriormente analizar, para que el programa decida qué hacer con ellos. Hay que saber que entre el 5 % y el 20 % de los mutantes son típicamente equivalentes [6].

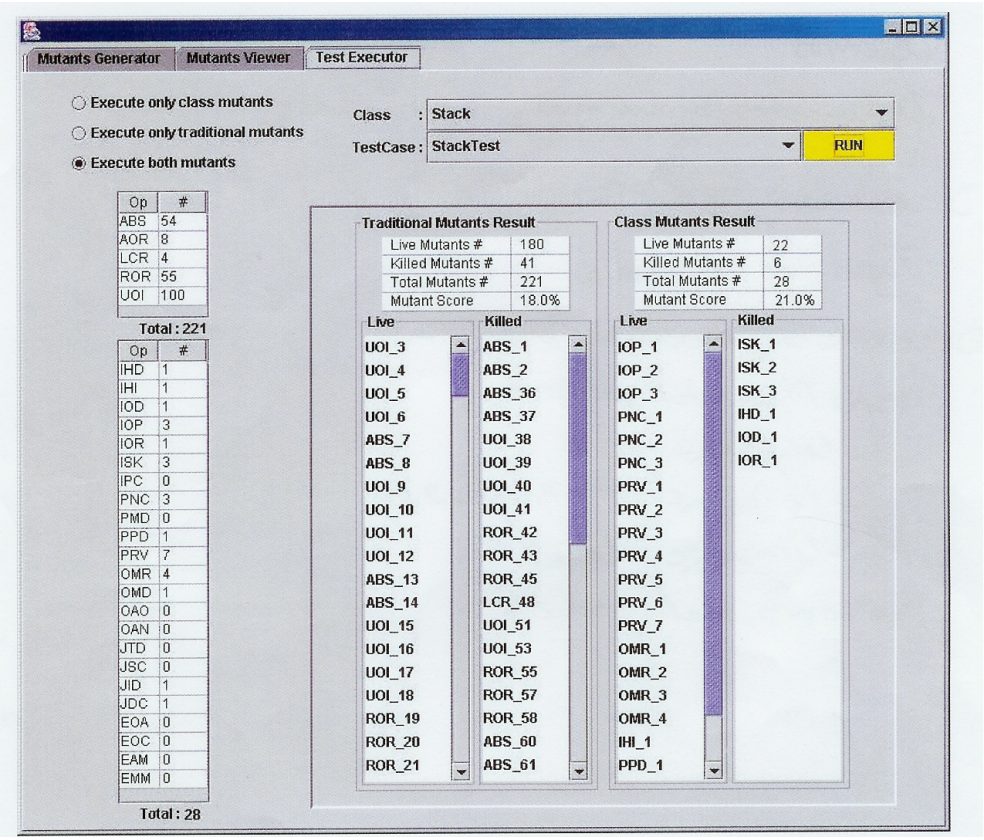

Figura 1.7: Captura μJava, tercera sección

Como ya hemos comentado, vamos a basarnos en μJava para realizar la aplicación de nuestro proyecto. De esta herramienta, escogeremos la mayor parte de las ideas y componentes ya que ésta obtiene la mayor parte de elementos que necesitamos para nuestro estudio. La visualización final de los resultados estadísticos sobre los mutantes vivos o muertos es algo que nos interesa bastante.

De igual modo, en esta última pantalla nos interesa la clasificación que realizan en la que se muestran el número de mutantes que han sido matados según el operador. Nosotros añadiremos otra columna en la que incluiremos el caso de prueba que mató al mutante.

Hablando de lo que es la visualización de los códigos, tanto original, como mutante, lo dejaremos para unos puntos más hacia delante. Y a la hora de escoger y seleccionar los ficheros u operadores, según nuestro gusto o lo que consideremos más adecuado, lo realizaremos como μJava o elaboraremos otro modo.

1.5.3. Matando mutantes con μClipse

μClipse es un plugin de Eclipse que proporciona un puente entre la API de μJava y la plataforma Eclipse [7].

μClipse es una reencarnación de la herramienta μJava pero desde un plugin de Eclipse. El proceso de mutación requiere una gran manipulación de un meta-lenguaje. Dado que los operadores mutantes actúan sobre el código fuente, los mutantes producidos, deben de ser compilados antes si se quiere que se ejecuten con los test de los casos de prueba. Además, los test mutantes requieren que el código fuente mutante no esté en lenguaje Java (classpath Java) si nosotros queremos comprobar si pueden matarse. Cada uno de estos pasos requieren medidas muy concretas de configuración en tiempo de ejecución, que son manejados por las configuraciones que vienen con la instalación de μClipse [7].

μClipse prevé avances para el sistema de μJava en las áreas de uso y compatibilidad. Las siguientes mejoras ayudan a explicar su motivación [7]:

- Gestión de clases
- Estructura de directorios de configuración
- Configuración en tiempo de ejecución
- Compatibilidad con JUnit TestCases
- Integridad en los resultados GUI-based

μClipse está basado en el API de μJava y algunas cosas que no están implementadas en μJava, no están añadidas a μClipse. Estos son los diferentes casos que hemos encontrado que causan problemas a la hora de generar mutantes [7]:

- Genéricos
- Enums
- Anotaciones
- Argumentos variables
- Importaciones estáticas

Además, encontramos que con esta versión de μClipse, no se puede iniciar una nueva configuración en tiempo de ejecución para generar mutantes o ejecutar pruebas sin la selección de un proyecto en el "Explorador de proyectos" de Eclipse [7].

1.5.4. μClipse paso a paso

No nos queremos meter en la configuración de μClipse ya que realmente sólo nos interesa de esta herramienta su entorno. De ésta sacaremos ideas, componentes o elementos que nos puedan resultar más útiles, estéticos o llamativos a la hora de elaborar nuestra interfaz.

Como hemos comentado, no vamos a meternos en profundidad, con lo que sólo vamos a ir mostrando cada una de las pantallas de las que se compone μClipse y las operaciones que se realizan en cada una.

En la primera captura que mostramos (pasos para la generación de mutantes), se llega después de escribir una serie de comandos e indicarle a la aplicación mediante "Run", que nos abra la pestaña donde escogeremos la ubicación de los mutantes, tanto para obtenerlos y trabajar con ellos, como para almacenar los resultados.

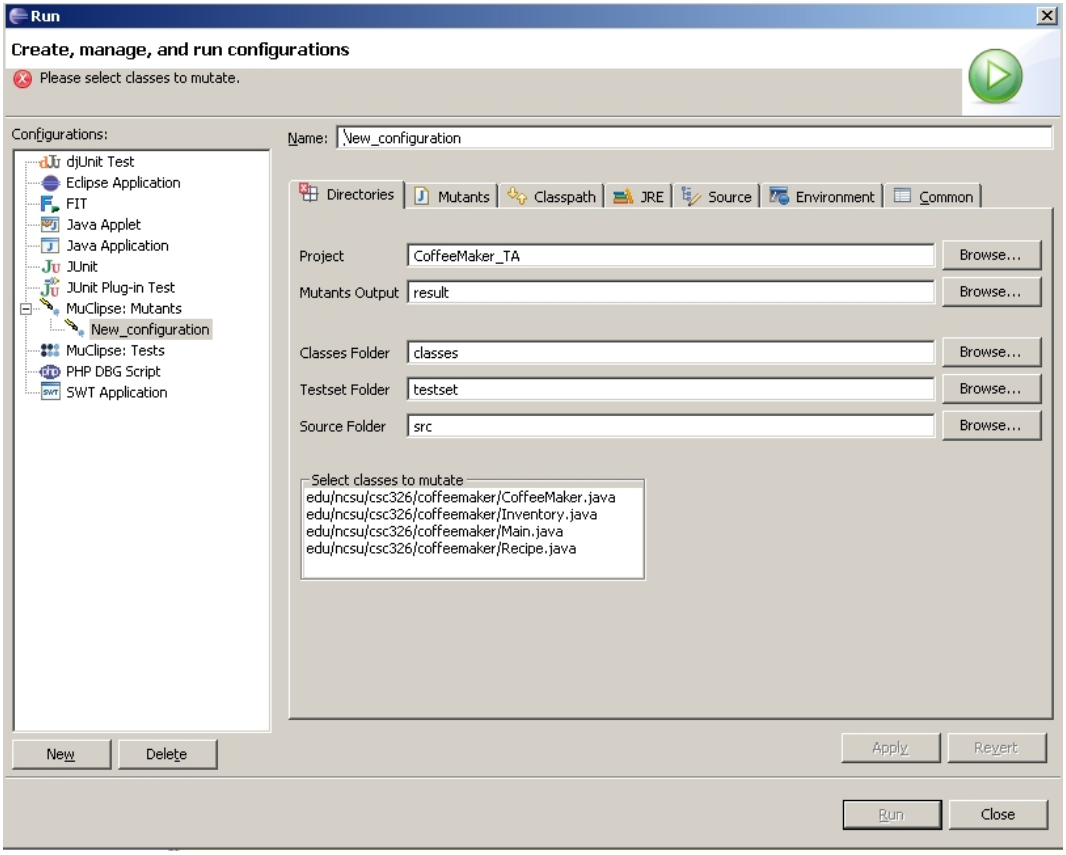

Figura 1.8: Captura μClipse, primera sección

En la segunda captura, que es la segunda pestaña que aparece en la imagen anterior, se trata de los operadores mutantes. En esta pestaña escogeremos los mutantes con los que queremos trabajar, crear los mutantes. Al igual que en μJava se nos facilitan una serie de botones con los que podemos seleccionar todos o ninguno de los "Traditional" o todos o ninguno de los "Class-level".

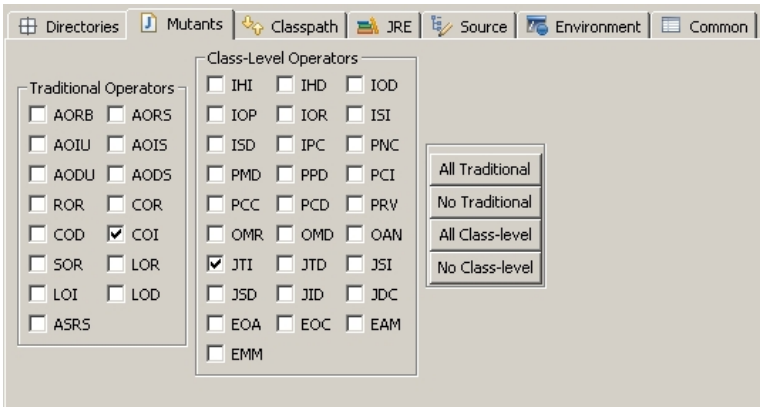

Figura 1.9: Captura μClipse, segunda sección

μClipse creará los mutantes que hemos especificado. Algunos de los operadores no producirán ningún mutante. Si esto es así, habrá que probar diferentes operadores. La consola nos mostrará un mensaje comunicándonos "¡Los mutantes han sido creados!" cuando el proceso se haya completado. Si quisiéramos visualizar los mutantes obtenidos, tendremos que proceder a verlos en la sección "Viendo los resultados" [8].

Ahora, mostraremos las pantallas para la ejecución de los casos de prueba. Volvemos a la pantalla inicial, donde escogemos los mutantes con los que vamos a trabajar y dónde queremos almacenar los resultados.

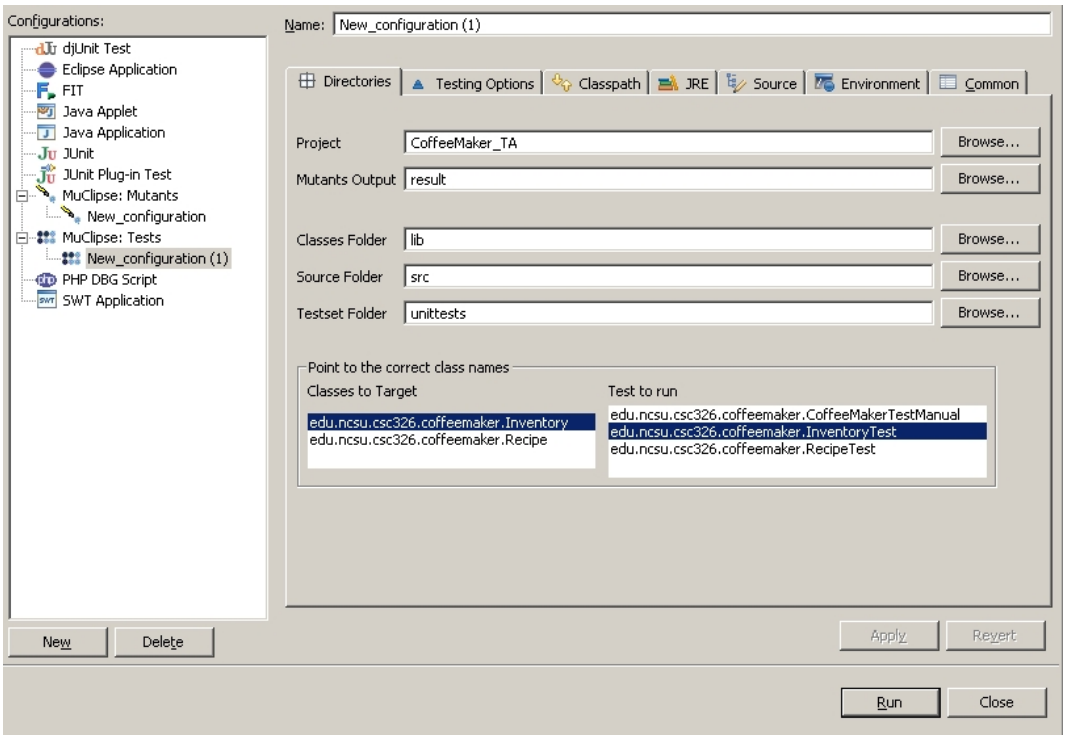

Figura 1.10: Captura μClipse, primera sección

La herramienta que nos ofrecen en μClipse para los casos de pruebas es la de "Time out period", que es para cuando queremos controlar el tiempo de los casos de pruebas (el tiempo de cada caso de prueba para ejecutarse). Si se produce un bucle infinito, se mata a un hilo de μClipse y cuenta como un fracaso, si no, se aumenta este valor. En el caso de que la ejecución esté llevando demasiado tiempo en ejecutarse, es recomendable disminuir este valor [8].

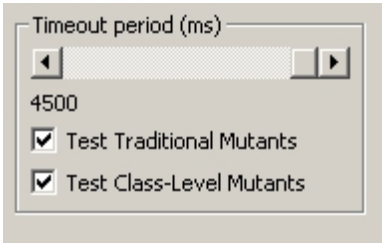

Figura 1.11: Captura μClipse, Time out period

Cuando se termine de especificar las opciones, ejecutamos. Se nos aparecerá en la consola una ventana de salida indicando los resultados de la prueba. En esta misma

ventana sale la puntuación de muertos y vivos. Podemos ver los resultados reflejados en el código dándole "Ver resultados y mutantes". Como podemos observar en la siguiente figura, aparecerá el código original, y podremos escoger el operador mutante para comparar los resultados.

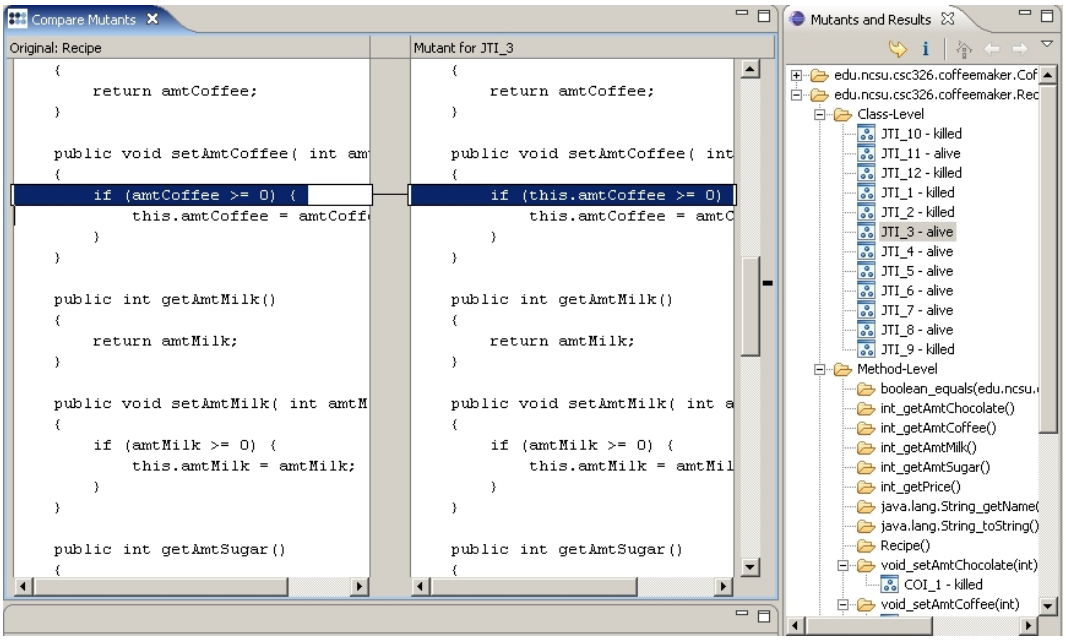

Figura 1.12: Captura μClipse, visualización de resultados

Ésta es una de las ideas o componentes que principalmente vamos a seleccionar de esta herramienta, ya que el resultado: los cambios en el código, quedan más visibles y se puede ir escogiendo el mutante tanto vivo como muerto para la comparación del código.

1.6. GAmera

Es un sistema de generación de mutantes para composiciones WS-BPEL. Para su implementación se han utilizado unos 26 operadores de mutación que se clasifican en cuatro categorías atendiendo al tipo de elemento sintáctico con el que se relacionan. Las categorías se identifican mediante letras mayúsculas: I (mutación de identificadores), E (mutación de expresiones), A (mutación de actividades) y X (mutación de condiciones excepcionales y eventos) [3].

Se han definido varios operadores de mutación dentro de cada categoría. Los operadores se identifican mediante tres letras mayúsculas: la primera es el identificador de la categoría, mientras que las otras dos indican el operador dentro de la categoría. Estos operadores de mutación modelan errores comunes que pueden cometer los programadores al implementar una composición WS-BPEL [3].

La siguiente figura muestra los tres componentes principales del sistema: el analizador, el generador de mutantes y el sistema de ejecución. El analizador recibe como entrada la definición del proceso original WS-BPEL y genera la información necesaria para el generador de mutantes. El generador de mutantes tiene dos componentes: un algoritmo genético en el que cada individuo representa a un mutante, y un conversor de individuos a mutantes. Este generador tiene como objetivo generar mutantes de calidad para composiciones de servicios WS-BPEL. Por último, el sistema de ejecución se encarga de ejecutar el programa original y los mutantes frente a los casos de prueba y comparar las salidas que producen [3].

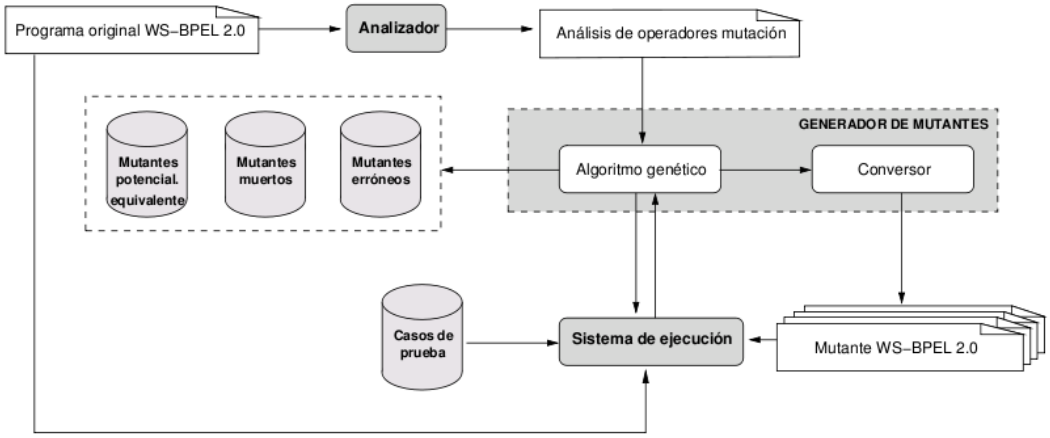

Figura 1.13: Componentes de GAmera

Los pasos de los que hemos hablado, se detallan en la sección del "Desarrollo general del proyecto", donde indicamos los ficheros que emplean, que se generan, las instrucciones con las que se llaman...

Capítulo 2 Desarrollo del Calendario

2.1. Fases de Desarrollo

A continuación se especifican y enumeran las distintas tareas que se han llevado a cabo para la elaboración del PFC. Estas tareas las representaremos mediante bloques y se pueden solapar en el tiempo y no se tiene que terminar completamente la elaboración de una para el comienzo de la siguiente ya que no son actividades críticas.

Gracias al "Diagrama de Gantt" se puede ver gráficamente y de una manera clara el solapamiento del que hablamos.

La elaboración total del proyecto ha tenido un coste en tiempo aproximado de 12 meses (desde Septiembre 2008 hasta Septiembre 2009) aunque se verá posteriormente que la dedicación al mismo no ha sido plena, debido a que el desarrollo de Koura ha estado compaginada con la finalización del quinto curso de la licenciatura Ingeniería en Informática.

1. En este primer bloque se estudia el lenguaje de programación que vamos a utilizar para implementar Koura. También las herramientas que vamos a utilizar a lo largo del desarrollo. Es un aprendizaje contínuo en el que cada vez que se cambiaban los requisitos, o al inicio del proyecto, teníamos que estudiar las posibilidades que nos ofrecían dichos lenguajes y herramientas.

2. El segundo bloque es más teórico porque aprendemos los conceptos de las mutaciones para llevarlos luego a cabo. También se contemplan los estudios realizados en otras herramientas (MuJava y MuClipse), relacionados con las mutaciones. No solo nos van a servir para la memoria, sino también para sacar algunos requisitos y para comprender el funcionamiento que se espera que tenga Koura en cada momento y solventar los posibles errores que nos puedan surgir.

3. El tercer bloque es ya la implementación de nuestro proyecto Koura, cada vez que se tiene una nueva idea para Koura se toca este bloque, requisitos, cambios, errores, diseño...

4. El cuarto bloque es para la recogida de requisitos. Obviamente este bloque tiene lugar al inicio del proyecto, pero debido al modelo en espiral que se ha seguido, cuando Koura estuvo casi por la mitad de su desarrollo no se terminaron de tener claros. Luego con una actualización de Mono, hubo algunos cambios que aparecen a mitad del desarrollo.

5. El quinto bloque es el diseño, en un principio se hizo un simil en el diseño de las herramientas estudiadas (MuJava y MuClipse), luego se nos propuso otro diseño más acorde a las necesidades del grupo de investigación, y por último se hicieron modificaciones en este último según la guía de estilos [**?**].

6. El sexto bloque es para la memoria, se empezó desde casi el principio del desarrollo del proyecto, luego debido a que había que tener terminado Koura para la presentación de las JISBD, se dejó apartada centrándose más en la implementación. Una vez terminada Koura se continuó con la memoria.

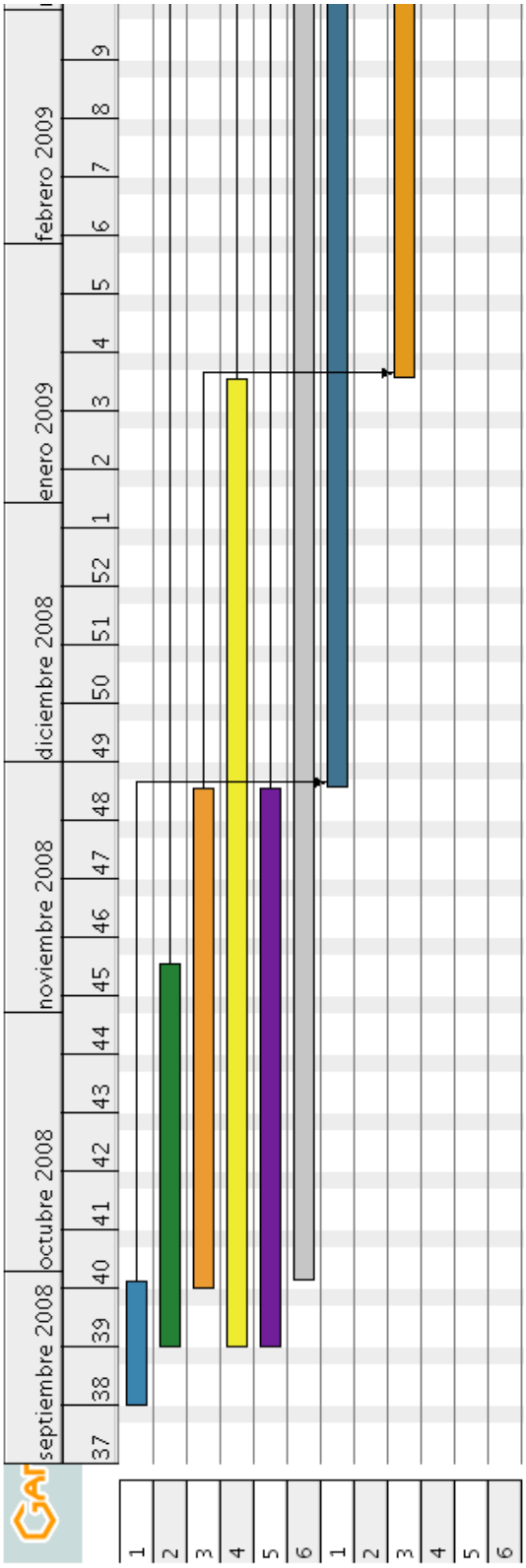

Figura 2.1: Diagrama de Gantt - Parte 1

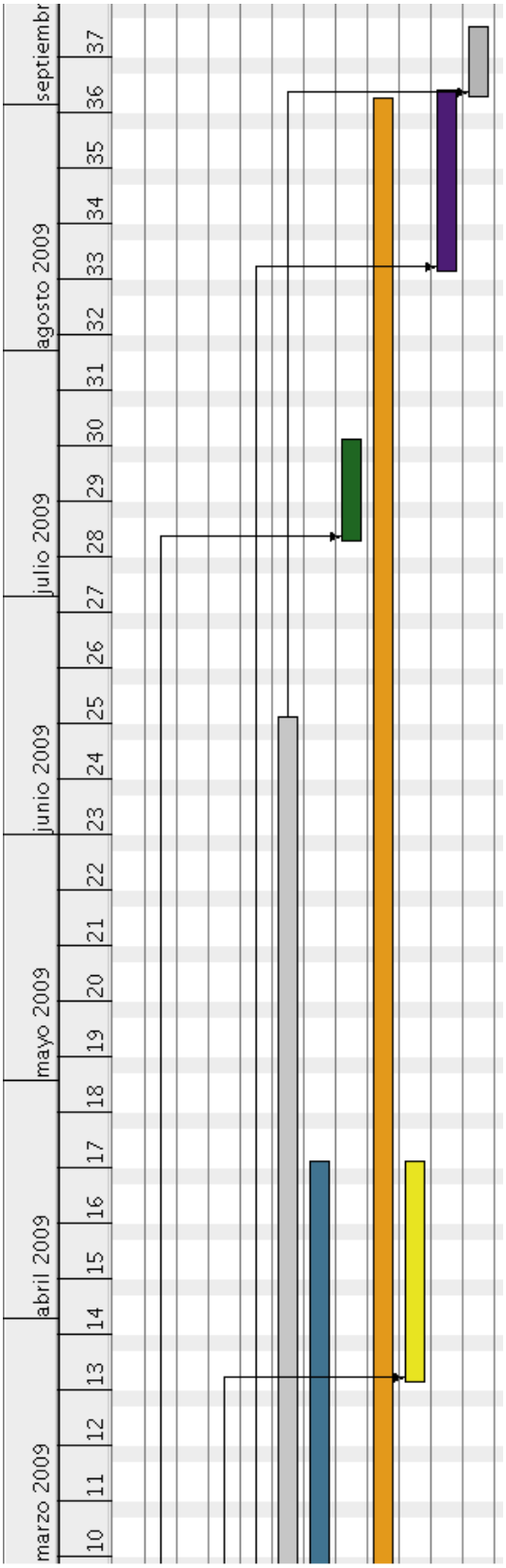

Figura 2.2: Diagrama de Gantt - Parte 2

Capítulo 3 Descripción General del Proyecto Fin de Carrera

3.1. Perspectiva del proyecto

Koura es una aplicación orientada a la investigación de los algoritmos mutantes con el lenguaje WS-BPEL. Su ámbito, a pesar de que WS-BPEL sea un lenguaje comercial, es hacia la investigación ya el objetivo de la aplicación es el estudio de los algoritmos mutantes.

3.1.1. Lenguaje C ♯

C ♯ (pronunciado "si sharp", o en español "ce sharp") es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO [5].

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (entre ellos Delphi) [5].

El símbolo ♯ viene de sobreponer "++" sobre "++" y eliminar las separaciones, indicando así su descendencia de C++ [5].

C ♯, como parte de la plataforma .NET, está normalizado por ECMA desde diciembre de 2001 (ECMA-334 "Especificación del Lenguaje C ♯"). El 7 de noviembre de 2005 salió la versión 2.0 del lenguaje que incluía mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables. El 19 de noviembre de 2007 salió la versión 3.0 de C ♯ destacando entre las mejoras los tipos implícitos, tipos anónimos y el LINQ (Language Integrated Query) [5].

Aunque C ♯ forma parte de la plataforma .NET, ésta es una interfaz de programación de aplicaciones; mientras que C ♯ es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado el que provee el Framework de DotGNU - Mono es que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX / Linux [5].

3.1.2. Proyecto Mono

Mono es el nombre de un proyecto de código abierto iniciado por Ximian y actualmente impulsado por Novell (tras su adquisición de Ximian) para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET según lo especificado por el ECMA [5].

Mono posee importantes componentes útiles para desarrollar software [5]:

- Una máquina virtual de lenguaje común de infraestructura (CLI) que contiene un cargador de clases, un compilador en tiempo de ejecución (JIT), y unas rutinas de recolección de memoria.
- Una biblioteca de clases que puede funcionar en cualquier lenguaje que funcione en el CLR (Common Language Runtime).
- Un compilador para el lenguaje C ♯, MonoBasic (la versión para mono de Visual Basic), Java y Python.
- El CLR y el Sistema de tipos común (CTS) permite que la aplicación y las bibliotecas sean escritas en una amplia variedad de lenguajes diferentes que compilen para byte code
- Esto significa por ejemplo, que si defines una clase que haga una manipulación algebraica en C ♯, esa clase puede ser reutilizada en cualquier lenguaje que soporte el "CLI". Puede crear una clase en C ♯, una subclase en C++ e instanciar esa clase en un programa en Eiffel.
- Un sistema de objetos único, sistema de hilos, bibliotecas de clases y sistema recolector de memoria pueden ser compartidos por todos estos lenguajes.
- Es un proyecto independiente de la plataforma. Actualmente Mono corre sobre Linux, FreeBSD, UNIX, Mac OS X, Solaris y plataformas Windows.

3.1.3. Lenguaje WS-BPEL 2.0

WS-BPEL es un lenguaje basado en XML que permite especificar el comportamiento de un proceso de negocio basado en interacciones con WS. Nació como combinación de WSFL (Web Service Flow Languaje de IBM, orientado a grafos y basado en el control de los links entre tareas) y XLANG (Web Services for Business Process Design de Microsoft, basado en un control de flujos con secuencias, condiciones y bucles, etc...) y ha evolucionado adquiriendo lo mejor de cada uno e intentando evitar las malas prácticas de los mismos.

La estructura de un proceso WS-BPEL se divide en cuatro secciones [1]:

1. Definición de relaciones con los socios externos, que son el cliente que utiliza el proceso de negocio y los WS a los que llama el proceso.
2. Definición de las variables que emplea el proceso.
3. Definición de los distintos tipos de manejadores que puede utilizar el proceso. Pueden definirse manejadores de fallos, que indican las acciones a realizar en caso de producirse un fallo interno o en un WS al que se llama. También se definen los manejadores de eventos, que especifican las acciones a realizar en caso de que el proceso reciba una petición durante su flujo normal de ejecución.
4. Descripción del comportamiento del proceso de negocio; esto se logra a través de las actividades que proporciona el lenguaje.

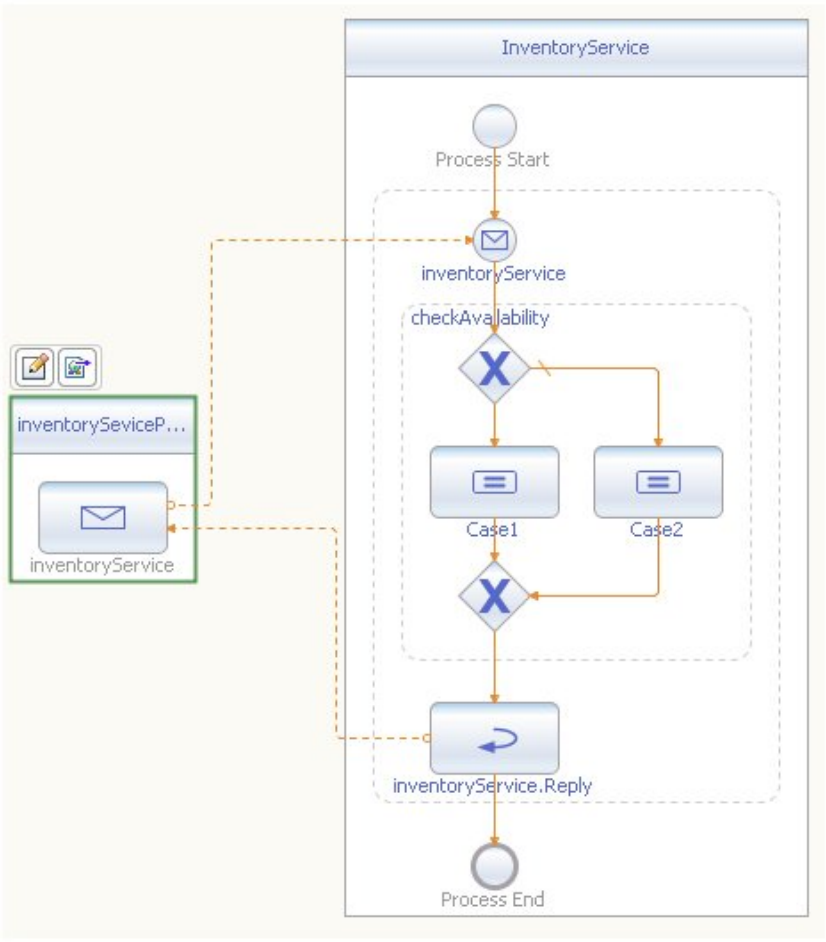

Figura 3.1: WS-BPEL 2.0

Todos los elementos definidos anteriormente son globales si se declaran dentro del proceso. Sin embargo, también existe la posibilidad de declararlos de forma local me-

diante el contenedor *scope*, que permite dividir el proceso de negocio en diferentes ámbitos [1].

Los principales elementos constructivos de un proceso WS-BPEL son las actividades, que pueden ser de dos tipos: básicas y estructuradas. Las actividades básicas son las que realizan una determinada labor (recepción de un mensaje, manipulación de datos, etc.). Las actividades estructuradas pueden contener otras actividades y definen una lógica de negocio [1].

A las actividades pueden asociarse un conjunto de atributos y un conjunto de contenedores. Estos últimos pueden incluir diferentes elementos, que a su vez pueden tener atributos asociados. Veamos un ejemplo [1]:

```
<flow> {Actividad estructurada}
 <links>  {Contenedor}
  <link name=''comprobarVuelo-A-reservarVuelo'' {Atributo}/> {Elemento}
 </links>
 <invoke name=''comprobarVuelo'' ... > {Actividad basica}
  <sources> {Contenedor}
   <source linkName=''comprobarVuelo-A-reservarVuelo'' {Atributo}/> {Elemento}
  </sources>
 </invoke>
 <invoke name=''comprobarHotel'' ... />
 <invoke name=''comprobarAlquilercoche'' ... />
 <invoke name=''reservarVuelo'' ... />
  <targets> {Contenedor}
   <target linkName=''comprobarVuelo-A-reservarVuelo''/> {Elemento}
  </targets>
 </invoke>
</flow>
```

Además, WS-BPEL permite realizar acciones en paralelo y de forma sincronizada. Un ejemplo de ello es la actividad *flow*, mediante la cual se puede especificar un conjunto de actividades que se van a realizar concurrentemente, así como las condiciones de sincronización entre ellas [1].

3.1.4. Lenguaje Perl

Perl, Lenguaje Práctico para la Extracción e Informe (Practical Extraction and Report Languaje) es un lenguaje de programación diseñado por Larry Wall creado en 1987. Perl toma características del C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, muchos otros lenguajes de programación [5].

Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script [5].

Perl es software libre y está licenciado bajo la Licencia Artística y la GNU General Public License. Existen distribuciones disponibles para la mayoría de sistemas operativos. Está especialmente extendido en Unix y en sistemas similares, pero ha sido portado a las plataformas más modernas (y otras más obsoletas) [5].

Características a tener en cuenta del lenguaje:

- La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas.
- Perl también toma características de la programación shell. Todas las variables son marcadas con un signo precedente (sigil). Los sigil identifican inequívocamente los nombres de las variables, permitiendo a Perl tener una rica sintaxis. Notablemente, los sigil permiten interpolar variables directamente dentro de las cadenas de caracteres (strings). Como en los shell, Perl tiene muchas funciones integradas para tareas comunes y para acceder a los recursos del sistema.
- Perl toma las listas del Lisp, hash (memoria asociativa) del AWK y expresiones regulares del sed. Todo esto simplifica y facilita todas las formas del análisis sintáctico, manejo de texto y tareas de gestión de datos.
- En Perl 5, se añadieron características para soportar estructuras de datos complejas, funciones de primer orden (p. e. clausuras como valores) y un modelo de programación orientada a objetos. Éstos incluyen referencias, paquetes y una ejecución de métodos basada en clases y la introducción de variables de ámbito léxico, que hizo más fácil escribir código robusto (junto con el pragma strict). Una característica principal introducida en Perl 5 fue la habilidad de empaquetar código reutilizable como módulos. Larry Wall indicó más adelante que "la intención del sistema de módulos de Perl 5 era apoyar el crecimiento de la cultura Perl en vez del núcleo de Perl".
- Todas las versiones de Perl hacen el tipado automático de datos y la gestión de memoria. El intérprete conoce el tipo y requerimientos de almacenamiento de cada objeto en el programa; reserva y libera espacio para ellos según sea necesario. Las conversiones legales de tipo se hacen de forma automática en tiempo de ejecución; las conversiones ilegales son consideradas errores fatales.

Perl se usa a menudo como un "lenguaje pegamento", ligando sistemas e interfaces que no fueron diseñados específicamente para interoperar; y para el "escarbado de datos", convirtiendo o procesando grandes cantidades de datos para tareas como por ejemplo crear informes. De hecho, estas fortalezas están íntimamente unidas. Su combinación hace a Perl una popular herramienta de propósito general para los administradores de sistemas, especialmente en programas pequeños que pueden ser escritos y ejecutados en una sola línea de comandos [5].

3.1.5. Scripts y líneas de comando

También hacemos empleo del lanzamiento de scripts para realizar en paralelo algunas operaciones que nos harán falta a lo largo del programa importando y exportando información del mismo para realizar estas operaciones y para obtener datos de las operaciones paralelas. Veamos con detenimiento qué es un script y sus características [5]:

Línea de comandos, Intérprete de mandatos, Intérprete de línea de mandatos, Intérprete de comandos, Terminal, Consola, Shell ó su acronimo en inglés CLI por Command line interface, es un programa informático que actúa como Interfaz de usuario para comunicar al usuario con el sistema operativo mediante una ventana que espera comandos textuales ingresados por el usuario en el teclado, los interpreta y los entrega al sistema operativo para su ejecución. La respuesta del sistema operativo puede ser mostrada al usuario en la misma ventana (dependiendo de los comandos). A continuación, la shell queda esperando más instrucciones [5].

Dada la importancia de esta herramienta, existe ya desde los comienzos de la computación. Existen para diversos sistemas operativos, diversos hardware, con diferente funcionalidad. Suelen incorporar características tales como control de procesos, redirección de entrada/salida, listado y lectura de ficheros, protección, comunicaciones y un lenguaje de órdenes para escribir programas por lotes o (scripts o guiónes) [5].

El guión o archivo de procesamiento por lotes (en inglés script) es un programa usualmente simple, que generalmente se almacena en un archivo de texto plano. Los guiones son casi siempre interpretados, pero no todo programa interpretado es considerado un guión. El uso habitual de los guiones es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. Por este uso es frecuente que los shells sean a la vez intérpretes de este tipo de programas [5].

3.1.6. Lenguaje LaTeX

Para la realización de esta memoria empleamos el lenguaje LaTeXque nos ofrece un documento con entorno agradable y elegante [5].

Es un lenguaje de marcado para documentos, y un sistema de preparación de documentos, formado por un gran conjunto de macros de TeX, escritas inicialmente por Leslie Lamport (LamportTeX) en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica creado por Donald Knuth. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeXes comparable a la de una editorial científica de primera línea. LaTeXes software libre bajo licencia LPPL [5].

LaTeXes un procesador de textos que está formado mayoritariamente por órdenes (macros) construidas a partir de comandos de TeX lenguaje «de bajo nivel», en el sentido de que sus acciones últimas son muy elementales pero con la ventaja añadida, en

palabras de Lamport, de «poder aumentar las capacidades de LaTeXutilizando comandos propios del TeX descritos en The TeXbook» . Esto es lo que convierte a LaTeXen una herramienta práctica y útil pues, a su facilidad de uso, se une toda la potencia de TeX. Estas características hicieron que LaTeXse extendiese rápidamente entre un amplio sector científico y técnico, hasta el punto de convertirse en uso obligado en comunicaciones y congresos, y requerido por determinadas revistas a la hora de entregar artículos académicos [5].

3.2. Funciones del producto

Koura se divide en cuatro secciones bien diferenciadas: análisis, generación y ejecución, resultados y ver mutantes. Describamos con profundidad cada una de estas partes.

3.2.1. Análisis

En el análisis, a partir de un archivo WS-BPEL y un directorio de trabajo, que nos servirá para ir almacenando todos los ficheros que se vayan generando a lo largo del estudio, ejecutamos la siguiente instrucción:

mutationtool analyze fichero.bpel

Esto nos dará un fichero que denominaremos “Nombreficherobpel.bpel.operadores_original” y se nos mostrará en pantalla aquellos operadores que se emplean en el caso de prueba que hayamos seleccionado. Este fichero, “Nombreficherobpel.bpel.operadores_original” tiene la siguiente estructura:

Nombre_operador Número_instrucciones Atributo

El “Nombre_operador” nos aparecerá una primera columna con los nombres de los operadores (por ahora 26, ya que se tienen perspectivas de seguir elaborando más), en una segunda columna aparece “Número_instrucciones”, que consiste en un número que nos indica el número de instrucciones donde se aplica (en el caso de que este valor sea 0 quiere decir que no se aplica en el ejemplo dicho operador) y finalmente “Atributo”, que es el atributo asociado al operador.

Aquí un ejemplo del fichero del que hablamos:

```
1  ISV 0 1
2  EAA 0 4
3  EEU 0 1
4  ERR 2 5
5  ELL 0 1
6  ECC 0 1
7  ECN 1 4
8  EMD 0 2
9  EMF 0 1
10 ACI 0 1
```

```
AFP 0 1
ASF 2 1
AIS 0 1
AIE 2 1
AWR 0 1
AJC 0 1
ASI 3 1
APM 0 1
APA 0 1
XMF 0 1
XRF 1 1
XMC 0 1
XMT 0 1
XTF 0 1
XER 0 1
XEE 0 1
```

Listado 3.1: Nombreficherobpel.operadores_original

En la aplicación se nos permite la opción de seleccionar, dentro de los operadores que se muestran como activos (lo que quiere decir que se pueden aplicar al ejemplo que estemos estudiando), aquellos con los que queremos trabajar y descartar aquellos que, aunque activos, no queremos estudiarlos. Esta selección se guardará en el fichero "Nombreficherobpel.bpel.operadores" y será este fichero el que emplearemos en las siguientes fases en vez del "Nombreficherobpel.bpel.operadores_original".

3.2.2. Generación y ejecución

Aquí se nos ofrecen dos opciones, el llamar al algoritmo genético con todos los operadores, o hacer una selección de éstos para trabajar con ellos.

En el caso de trabajar con todos, la generación y ejecución se realizan mediante la siguientes instrucciones:

mutationtool apply fichero.bpel OPERADOR INSTRUCCION ATRIBUTO

Esto genera para cada mutante seleccionado, OPERADOR, INSTRUCCION y ATRIBUTO un ficheromutante.bpel. Estos ficheros están almacenados dentro del directorio de trabajo en una carpeta llamada "mutantes"

mutationtool run ficheroprueba.bpts fichero.bpel

Nos da un fichero fichero.out que nos servirá para la siguiente instrucción:

mutationtool compare ficheroprueba.bpts fichero.bpel fichero.out ficheromutante.bpel

Para cada mutante que se nos haya generado nos dará un fichero salida que nos indicará mediante los valores 0, 1 y 2 si el test que viene denominado según su posición en la

fila que contiene el fichero lo ha matado, ha dado error o bien sigue vivo. Estos ficheros se almacenan en nuestro directorio de trabajo en una carpeta denominada “Salida”.

El fichero se compone de una simple línea con tantas columnas como test se realicen. Si todas las columnas están a 0 quiere decir que el mutante está vivo, si existe algún 1 en alguna de las columnas, estamos hablando de un mutante muerto por el test que ocupa el número de la columna que tiene el número 1 y si hay un 2 es que ha dado error.

Aquí un ejemplo de los ficheros de salida:

```
0 0 0 0 0
```

Listado 3.2: Salida mutante vivo

```
0 0 0 1 1
```

Listado 3.3: Salida mutante muerto

```
2 2 2 2 2
```

Listado 3.4: Salida mutante erróneo

En el caso que queramos trabajar con una selección de operadores, la generación y ejecución no será realizada en varios pasos, se hará con la siguiente instrucción:

gamera seed file gamera.conf

Donde seed es un número que representa la semilla con la que vamos a trabajar y gamera.conf el nombre del fichero de donde leeremos unos valores que tendremos que introducir por pantalla en Koura, estos son (lo vamos a ver con un fichero de ejemplo):

```
TAMPOBLACION 5
NUMTEST 5
NUMGENERACIONES 5
PMUTACION 0.3
PCRUCE 0.7
PSUSTITUIR 1
PNUEVO 0.3
BPELORIGINAL 1 /home/lo/GAmera/LoanApprovalRPC/loanApprovalProcess.bpel.original
BPELOUT 0 LoanApprovalRPC/loanApprovalProcess.out
TESTSUITE 1 LoanApprovalRPC/loanApprovalProcess.bpts
SALIDAGAMERA 1 generaciones5_5_0,7_0,3.dat
ESTADISTICAS 1 estadisticas5_5_0,7_0,3.dat
```

```
13 DIFERENTES 0
14 BDMUTANTES 1 mutantes-generados5_5_0,7_0,3.dat
15 PMUTANTES 0.2
16 DIRECTORIOSALIDA 1 /home/lo/GAmera/LoanApprovalRPC/LoanAppAG
17 ANALIZADOR 1 /home/lo/GAmera/LoanApprovalRPC/LoanAppAG/loanApprovalProcess.
       bpel.operadores
```

Listado 3.5: gamera.conf

Al igual que con la otra opción, se nos van a generar una serie de ficheros WS-BPEL mutantes que se almacenarán en una carpeta denominada "mutantes" dentro del directorio de trabajo.

Estos dos métodos para la generación y ejecución de mutantes nos dan una serie de ficheros salida, además de los ya comentados. En ambos casos se nos va a devolver un fichero que se llama "mutantes-generadosXX_XX_XX_XX.dat", el fichero contiene cinco columnas, veamos sus significados con un ejemplo:

```
1 21    1    1    2    -1
2 7     1    2    0    -1
3 12    2    1    2    -1
4 17    2    1    1    0
```

Listado 3.6: mutantes-generados.dat

1. Número del operador
2. Número de la instrucción
3. Número del atributo
4. Estado del mutante que obtenemos con dicho operador, instrucción y atributo
5. Número del test que mató al mutante (en el caso de que en la columna anterior aparezca un uno, en caso contrario, el valor del test será siempre -1)

Sólo en el caso de que generemos una selección de mutantes obtenemos los ficheros "generacionesXX_XX_XX_XX.dat" y "estadisticasXX_XX_XX_XX.dat", aquí un ejemplo del contenido de sendos ficheros:

```
1 seed    0
2 minimaxi        1
3 number_of_generations 5
4 generations_to_convergence      20
5 convergence_percentage 0.99
```

```
crossover_probability    0.7
mutation_probability     0.3
population_size 5
score_frequency 10
flush_frequency 100
record_diversity         0
score_filename  gamera_estadisticas.dat
 select_scores     255
number_of_best 1
replacement_percentage 1
replacement_number     5
----------GENERACION 0----------
7       1       2       1       5
12      2       1       1       5
21      1       1       1       5
17      2       1       0       4
21      1       1       0       0
```

Listado 3.7: generaciones.dat

```
0       3.8     5       0       2.16795 -1
```

Listado 3.8: estadisticas.dat

Para observar toda esta información desde Koura, símplemente nos tendremos que dirigir a la pestaña "Resultados", donde según nuestra opción a la hora de trabajar con todos los mutantes o con una sección de ellos, se nos mostrarán unos ficheros u otros.

3.2.3. Resultados

En esta sección podemos visualizar los resultados de todos aquellos análisis que hayamos hecho con anterioridad seleccionando el fichero WS-BPEL con el que trabajamos y el directorio de trabajo donde almacenamos dicha información. Si venimos de realizar un estudio, ya se verán por pantalla el fichero WS-BPEL y el directorio de trabajo que estamos utilizando, luego simplemente tendremos que hacer clic en "Calcular" y se nos mostrará en pantalla un resumen de los resultados obtenidos, véanse:

```
tiempo 437,31
Num_mutantes_totales 22
Num_mutantes_generados 22
Num_mutantes_vivos 4
Num_mutantes_muertos 15
Num_mutantes_error 3

```

```
8
9   Opr        |  ERR |  ECN |  ASF |  AIE |  ASI |  XRF
10  Viv        |   1  |   3  |   0  |   0  |   0  |   0
11  Mts        |   9  |   1  |   1  |   2  |   2  |   0
12  Err        |   0  |   0  |   1  |   0  |   1  |   1
13  Gen        |  10  |   4  |   2  |   2  |   3  |   1
14
15
16
17  Numero del Test   | 0 | 1 | 2 | 3 | 4
18  Mutantes muertos  | 0 | 12 | 1 | 0 | 2
```

Listado 3.9: resumen.dat

- Tiempo (segundos)
- Número de mutantes generados
- Número de mutantes totales
- Número de mutantes vivos
- Número de mutantes muertos
- Número de mutantes erróneos

Menos el valor del tiempo, el resto de valores los calculamos del fichero "mutantes-generados", recorriéndolo y contabilizando según lo que necesitemos. El tiempo se calcula internamente en Koura, activándose el cronómetro cuando comienza la generación y ejecución y parándose el mismo cuando finalizan los procesos. Luego nos aparece un resumen detallado de cada uno de los operadores que intervienen en el estudio indicándonos (estos valores también se toman de "mutantes-generados"):

- Nombre
- Número de mutantes que permanecen vivos
- Número de mutantes que ha matado
- Número de mutantes generados con dicho operador
- Número de errores que se han producido

Y finalmente aparece otro resumen de cada uno de los test que se han realizado indicándonos (idem):

- Número del test
- Número de mutantes que ha matado

Estos resultados se pueden contemplar de igual modo en el fichero "resumen.dat" que aparece junto con el resto de ficheros indicados anteriormente en el directorio de trabajo. También generamos un fichero XML denominado "tablas.xml" en el que también se almacenan todos los datos que aparecen en la pantalla de Koura pero claro está, con un formato diferente por si en un futuro queremos trabajar con estos datos.

```
<?xml version="1.0" encoding="utf-8"?>
<Ejecucion>
  <Operadores>
    <Operador>
      <Nombre>ERR</Nombre>
      <Vivos>1</Vivos>
      <Muertos>9</Muertos>
      <Error>0</Error>
      <Generados>10</Generados>
    </Operador>
    <Operador>
      <Nombre>ECN</Nombre>
      <Vivos>3</Vivos>
      <Muertos>1</Muertos>
      <Error>0</Error>
      <Generados>4</Generados>
    </Operador>
    <Operador>
      <Nombre>ASF</Nombre>
      <Vivos>0</Vivos>
      <Muertos>1</Muertos>
      <Error>1</Error>
      <Generados>2</Generados>
    </Operador>
    <Operador>
      <Nombre>AIE</Nombre>
      <Vivos>0</Vivos>
      <Muertos>2</Muertos>
      <Error>0</Error>
      <Generados>2</Generados>
    </Operador>
    <Operador>
      <Nombre>ASI</Nombre>
      <Vivos>0</Vivos>
      <Muertos>2</Muertos>
      <Error>1</Error>
      <Generados>3</Generados>
    </Operador>
    <Operador>
      <Nombre>XRF</Nombre>
```

```
41          <Vivos>0</Vivos>
42          <Muertos>0</Muertos>
43          <Error>1</Error>
44          <Generados>1</Generados>
45        </Operador>
46      </Operadores>
47      <Tests>
48        <Test>
49          <NumTest>0</NumTest>
50          <MutMuertos>0</MutMuertos>
51        </Test>
52        <Test>
53          <NumTest>1</NumTest>
54          <MutMuertos>12</MutMuertos>
55        </Test>
56        <Test>
57          <NumTest>2</NumTest>
58          <MutMuertos>1</MutMuertos>
59        </Test>
60        <Test>
61          <NumTest>3</NumTest>
62          <MutMuertos>0</MutMuertos>
63        </Test>
64        <Test>
65          <NumTest>4</NumTest>
66          <MutMuertos>2</MutMuertos>
67        </Test>
68      </Tests>
69      <Tiempo>437,31</Tiempo>
70      <Mutantes>
71        <Total>22</Total>
72        <MGenerados>22</MGenerados>
73        <MVivos>4</MVivos>
74        <MMuertos>15</MMuertos>
75        <MErrores>3</MErrores>
76      </Mutantes>
77    </Ejecucion>
```

Listado 3.10: tablas.xml

3.2.4. Ver mutantes

Finalmente, para comparar las diferencias de forma visual entre los mutantes y el fichero original, tenemos este último proceso. Dado un fichero WS-BPEL original que compararemos y el directorio de trabajo, iremos escogiendo según el estado del operador (vivo, muerto o error), el operador que cumple estas condiciones, la instrucción y

atributo asignados a tal operador. De este modo ya tenemos el mutante y el original a visualizar.

Mediante un script de Perl, que hemos modificado, generamos un xml según nuestras necesidades que nos va a diferenciar el código del que hay que actualizar en el fichero original, del que hay que actualizar en el mutante, dejando intacto el código que es igual.

Para conseguir esto implementamos una nueva opción en el script de Perl para generar ficheros XML mediante la opción -X que nos diera las diferencias a través de un fichero que nosotros le indicásemos. El script de Perl tenía la opción de generar un fichero HTML con las diferencias, hemos adaptado esta opción a nuestras necesidades.

Como el script compara ficheros XML no hay que indicarle que genere nigún tipo de fichero diferente ya que tiene por defecto crear un fichero XML. Luego tuvimos que indicarle que antepusiera a las líneas diferentes del fichero mutante la cadena "#r" y a las líneas diferentes del fichero original "#v", de tal modo podemos diferenciar qué línea corresponde a cada fichero. El resto de indicaciones para crear el fichero de diferencias XML son las mismas que las que emplea el script de Perl con la opción -H para ficheros HTML, así que sólo tuvimos que indicarle que tuviese encuentra la opción seleccionada por el usuario (en nuestro caso -X), para que realizase las mismas operaciones que con la opción -H.

Para no tocar los originales, de los ficheros original WS-BPEL y mutante, nos vamos a crear nuevos ficheros con el resultado (fichero XML) que nos dió el script de Perl. Estos ficheros quedarán almacenados en un directorio nuevo dentro del directorio de trabajo denominado "Comparativas".

La llamada que realizamos para que se ejecute el Perl según nuestra opción implementada es la siguiente:

perl xmldiff.pl -X ficheroOriginal.bpel ficheroMutante.bpel >diferencia.xml

El fichero que nos da como resultado el script de Perl, es el fichero original que estamos comparando con algunas modificaciones. Se nos señalan con "#v" las líneas que hay que modificar en el fichero original para que se parezca al mutante y con "#r" aquellas líneas a modificar en el fichero mutante para que se parezca al original.

De este modo vamos leyendo del fichero con las diferencias y vamos escribiendo en los nuevos ficheros que nos servirán para la visualización en Koura, las líneas que no son diferentes en ambos ficheros y las líneas diferentes según la etiqueta o cadena correspondiente.

Una vez que tenemos estos ficheros y en Koura se seleccione la opción de visualizarlos para compararlos, se irán recorriendo ambos ficheros y cuando se lean las etiquetas "#v" en el fichero original coloreará esa línea en verde y cuando se lean las etiquetas "#r" en el fichero mutante, se coloreará esa línea en rojo. Esto lo conseguimos porque nos hemos creado unos "Tags" dentro de Koura que cuando lee las etiquetas de las que

hemos hablado, va coloreando las líneas en rojo o en verde (según la etiqueta) en el buffer asociado al área de texto donde se van a reflejar los ficheros a comparar.

Finalmente destacar que una de las veces que estábamos haciendo pruebas, comprobando que los operadores daban los resultados esperados, obtuvimos una incongruencia. Esto era debido a que el fichero mutante que se generaba, cuando encontraba en el fichero original el símbolo “>”, en el mutante escribía “gt;” o viceversa. Luego creamos un filtro en el que siempre que se encontrase “gt;” escribiera el símbolo “>”, de tal modo que así siempre tendríamos escrito tanto en el fichero original como en el mutante lo mismo, el símbolo “>”. Este filtro obviamente, se le pasa a los ficheros antes de realizar la comparativa (la llamada al script de Perl).

3.3. Características de los usuarios

Como ya hemos comentado, Koura es un proyecto orientado a la investigación, luego los usuarios de esta aplicación serán aquellos que necesiten trabajar con algoritmos mutantes en el lenguaje BPEL.

Los usuarios podrán encontrarse con multitud de funcionalidades para trabajar en este ámbito: análisis, generación y ejecución total o selectiva de mutantes, visualización de resultados (se tienen en cuenta una gran variedad de parámetros para obtener valores que abarquen todas las necesidades, en cuanto resultados, que el usuario pueda requerir) y finalmente la comparativa visual de cada uno de los mutantes con el fichero original.

Aunque el manejo de Koura no es complicado y menos si contamos con el manual de usuario (véase el capítulo 7), bien es cierto que el interpretar los resultados y tener conocimiento de lo que se está haciendo en cada momento, nos limita más a la hora de determinar los usuarios de Koura.

Tal y como se comentó en la introducción, este proyecto está elaborado para que el grupo de investigación de la Universidad de Cádiz SPI&FM automatice las tareas que venían realizando y para ayudarles a la hora de la visualización de resultados y comparativas de mutantes añadiéndoles mediante la aplicación que estamos tratando nuevas funcionalidades.

Por lo que podemos concluir, que aunque Koura pueda ser utilizado por cualquier persona, los más indicados serán los investigadores que estudien el ámbito de los algoritmos mutantes, en especial con el lenguaje WS-BPEL. Los principales usuarios de Koura serán el grupo de investigación SPI&FM de la Universidad de Cádiz.

3.4. Restricciones generales

La restricción principal viene en paralelo con las características de los usuarios. Aquellos usuarios que no tengan conocimientos sobre los algoritmos mutantes con el

lenguaje de programación WS-BPEL, aunque puedan hacer uso de Koura, no van a poder interpretar los resultados.

Bien es cierto que los ficheros que empleamos para ejecutar Koura, son unos archivos que no posee cualquier persona. Los poseen aquellas personas que estén investigando en el ámbito que estamos considerando.

Considerando que poseemos los conocimientos y los ficheros necesarios para manejar Koura, en cuanto el sistema operativo a emplear, podemos utilizar cualquier distribución Linux con los paquetes que describimos en la siguiente sección de "Suposiciones y dependencias". De hecho se hizo un estudio de campo, en el que probábamos Koura en el sistema operativo "Suse 11" y "Ubuntu 9 .04" y tras realizar una serie de cambios y comprobaciones, se consiguió adaptar Koura para que funcionase en ambos sistemas sin tener que realizar ninguna operación adicional. Luego el sistema operativo (Linux) no es un problema a la hora de ejecutar Koura.

Con respecto a las restricciones con los paquetes o aplicaciones que necesitamos hablamos en la siguiente sección de manera más detallada.

3.5. Suposiciones y dependencias

La principal dependencia que afecta a Koura viene dada por *GAmera*. GAmera tiene que ejecutarse bajo un entorno Sun Java 5. Casi todo funciona con OpenJDK 6, pero al ejecutar GAmera se producen unos fallos que no ocurren al usar Sun Java 5, por lo que se optó por dejarlo de ese modo.

También Koura se ve afectada por GAmera en:

- El guión de gestión de ActiveBPEL usa los servicios web de administración de ActiveBPEL, y para ello utiliza la herramienta curl.
- Tomcat 5.5
- ActiBPEL 4.1
- BPELUnit
- Gestor de mutantes WS-BPEL

En lo que concierne a la aplicación Koura, tenemos las siguientes dependencias:

- Como hemos empleado el lenguaje de programación C♯, Koura es dependiente de mono 2.0 (última versión en la actualidad). Empezamos con una versión diferente de mono cuando empezamos a implementar Koura y a la hora del cambio nos presentó algunos problemas pero fue en lo referente a la interfaz. Estos se solventaron y consideramos que no mostrarán problemas en un futuro con las posteriores versiones.

- También utilizamos scripts de Perl, con los que tampoco consideramos que haya ningún problema a la hora de las actualizaciones.

Capítulo 4 Desarrollo del Proyecto

La elaboración de un producto software de calidad requiere la aplicación, durante todo su desarrollo, de una metodología. Aunque no existe una definición estricta del concepto *metodología de desarrollo*, ésta puede considerarse como un conjunto de pasos y procedimientos que deben seguirse para desarrollar un producto software.

Para el desarrollo de este proyecto se han seguido las recomendaciones de **Métrica Versión 3**: Metodología de Planificación, Desarrollo y mantenimiento de sistemas de información, propuesta por el Ministerio de Administraciones Públicas.

METRICA Versión 3 ha sido concebida para abarcar el desarrollo completo de Sistemas de Información sea cual sea su complejidad y magnitud, por lo tanto su estructura responde a desarrollos máximos y debe adaptarse y dimensionarse en cada momento de acuerdo a las características particulares de cada proyecto. En este capítulo se incluyen los modelos considerados más relevantes del proceso de Desarrollo de Sistemas de Información, que incluye los siguientes subprocesos:

1. Estudio de Viabilidad del Sistema (EVI)
2. Análisis del Sistema de Información (ASI)
3. Diseño del Sistema de Información (DSI)
4. Construcción del Sistema de Información (CSI)
5. Implantación y Aceptación del Sistema (IAS)

4.1. Estudio de la Viabilidad del Sistema

4.1.1. Establecimiento del alcance del sistema

Debemos estudiar el alcance de la necesidad planteada por el cliente o usuario; en nuestro caso, el Proyecto fue planteado por mi tutor para que a la vez entrase como colaboradora en el grupo de investigación de la Universidad de Cádiz SPI&FM.

Koura se plantea como una aplicación necesaria para las actividades que se desarrollan en el grupo de investigación e incluso un avance a la hora de la obtención de resultados. Se ha presentado un artículo en las JISBD con la inclusión de Koura que ha sido aceptado y en la presentación de Septiembre del 2009 va incluida la herramienta.

4.1.2. Estudio de la situación actual y predefinición de requisitos

Actualmente el algoritmo genético y todos los estudios que se realizan con él se hacen manualmente, con lo que es muy tedioso el tener que organizar los casos de estudio, acordarse de las instrucciones, acordarse de los estudios realizados, escribir reiteradamente una serie de instrucciones, etc.

Con Koura solventamos todas estas tareas y añadimos nuevas funcionalidades que van a permitir al usuario tener nuevos resultados, junto con los que ya se obtenían de manera manual, pero de una forma más organizada.

Como requisitos: además de tener instalados todas las aplicaciones y dependencias ya comentadas, hay que tener ubicados todos los ficheros de pruebas necesarios para los estudios de las mutaciones, y especialmente tener los conocimientos necesarios para poder interpretar los resultados que nos ofrece la aplicación.

A la hora de ejecutar Koura, éste se situará en el directorio de GAmera. Dentro del directorio de GAmera se encontrarán cada uno de los directorios de los casos de prueba para realizar los estudios de las mutaciones (con todos los ficheros necesarios). Dentro de estos casos de prueba, se crearán los directorios de trabajo de Koura, que estarán ordenados según la organización que queramos darles mediante el nombre que les demos.

4.1.3. Estudio y valoración de alternativas de solución

Antes incluso de valorar las diferentes alternativas a Koura, se estudió las necesidades dentro del grupo de investigación de la Universidad de Cádiz SPI&FM. Se vio la necesidad de automatizar y ampliar algunas tareas de las que se estaban realizando, y ahí nació la aplicación Koura.

El nombre de Koura viene de caparazón en japonés. El motivo de su nombre es por la aplicación GAmera, cuyo significado proviene de una especie de monstruo tortuga en japonés. Como Koura trabaja a un nivel más alto que GAmera, se le denominó "caparazón" ya que de una forma u otra engloba a la aplicación de GAmera.

Una vez que se vieron las necesidades del grupo de investigación, se plantea el lenguaje de programación en el que implementar Koura.

Debido a que GAmera está implementado en Java se plantea esta opción, pero nos vemos limitados por la versión y optamos por seguir mirando más posibilidades.

Como ya teníamos conocimientos de las librerías Wx-Widgets, también se nos planteaba esta opción. Pero queríamos estudiar otros lenguajes de programación, con lo que seguimos buscando.

Finalmente, nuestra opción fue la de escoger el lenguaje de programación C♯, que está en auge y se asemeja bastante al C++.

Dentro de Koura necesitamos de otra herramienta para las diferencias entre mutantes (algo que se comenta también posteriormente), para lo cual se nos planteó en un principio la aplicación xmldiff, pero tras trabajar con ella observamos que nos iba a dar problemas en futuras versiones, así que tras observar otras herramientas, escogimos Perl que posee un script que nos ofrecía los mismos o incluso mejores resultados, hablando para las necesidades de Koura, que la aplicación xmldiff.

4.2. Análisis del Sistema

El propósito de este proceso es conseguir la especificación detallada del sistema de información a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso de Diseño del Sistema de Información.

Como ya hemos comentado MÉTRICA Versión 3 cubre tanto desarrollos estructurados como orientados a objetos y las actividades de ambas aproximaciones están integradas en una estructura común aunque presenta alguna actividad exclusiva para cada tipo de desarrollo.

Para el desarrollo de un proyecto es muy importante realizar un modelado tanto de análisis y diseño para obtener un conjunto de modelos que nos permitan especificar los requisitos, la estructura y el comportamiento del sistema.

Comenzaremos nuestro análisis del sistema realizando el modelado de casos de uso. Entendemos un caso de uso como una descripción que especifica un comportamiento deseado del sistema. Los casos de uso describen qué hace el sistema representando los requisitos funcionales.

Describiremos los casos de uso de nuestro proyecto ayudándonos de representaciones gráficas, diagramas de casos de uso y de una descripción del mismo.

Los casos de uso son una técnica de especificación de requisitos válida tanto en desarrollos estructurados como en orientación a objetos, aunque en este último caso se propone como técnica obligatoria al ser necesaria como referencia a lo largo de todo el ciclo de vida. En esta tarea se elabora el modelo de casos de uso, según las normas y estándares de la organización, identificando:

- Actores.
- Casos de uso.
- Breve descripción de cada caso de uso.

Esta descripción incluirá el nombre del caso de uso, actores que intervienen en él, precondiciones, postcondiciones, escenario principal y los posibles escenarios alternativos.

Los actores del sistema, son aquellos que intervienen en la aplicación. Entendemos por escenario principal, el flujo normal que seguirá la aplicación para este caso de uso y entendemos por escenarios alternativos, aquellas variantes que se pueden dar durante el transcurso del caso de uso (errores, otras opciones, etc).

4.2.1. Modelos de Casos de Uso

Empezaremos agrupando los diferentes casos de uso y seguidamente descenderemos en abstracción hasta los casos de uso más concretos.

4.2.2. Casos de uso asociados a los requisitos funcionales

Los casos de uso son una técnica de especificación de requisitos válida tanto en desarrollos estructurados como en orientación a objetos, aunque en este último caso se propone como técnica obligatoria al ser necesaria como referencia a lo largo de todo el ciclo de vida. En esta tarea se elabora el modelo de casos de uso, según las normas y estándares de la organización, identificando:

- Actores.
- Casos de uso.
- Breve descripción de cada caso de uso.

Los productos obtenidos en la tarea Determinación del Alcance del Sistema, son tomados como referencia durante la obtención de requisitos, de forma que todos los requisitos especificados se encuentren dentro del ámbito del sistema de información.

4.2.3. Diagramas de los Casos de Uso

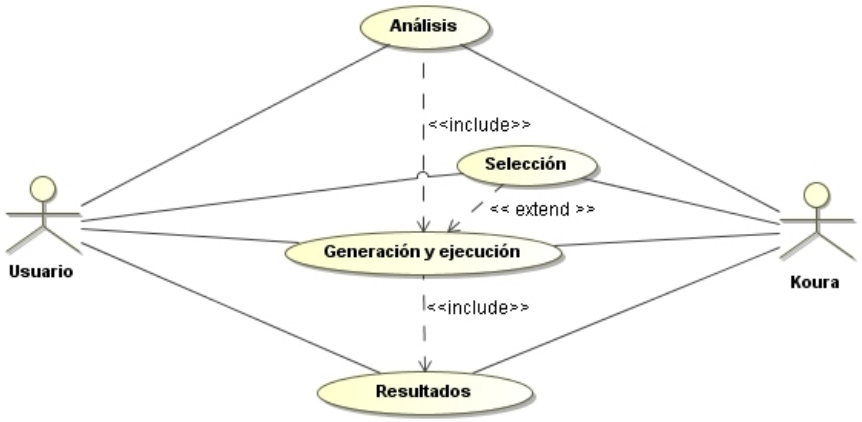

Figura 4.1: Diagrama general de los casos de uso

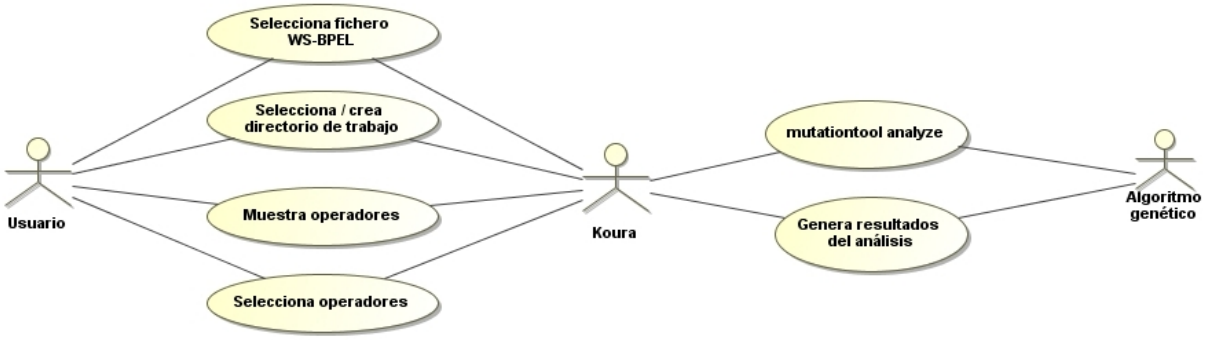

Figura 4.2: Diagrama del caso de uso "Análisis"

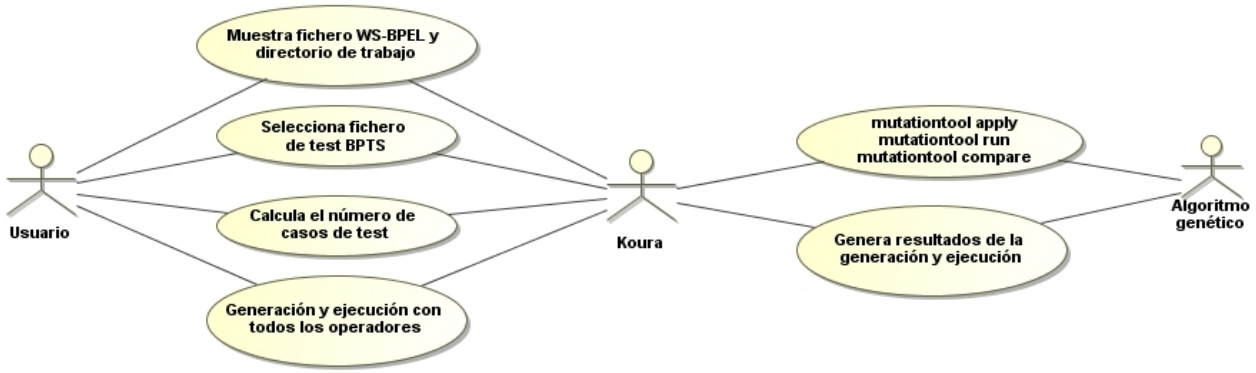

Figura 4.3: Diagrama del caso de uso "Generación y Ejecución - Todos"

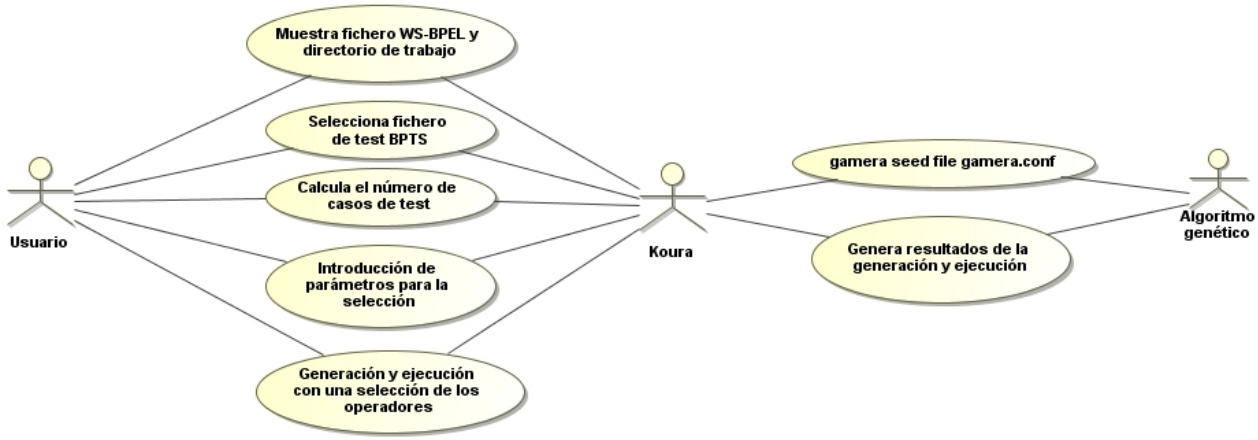

Figura 4.4: Diagrama del caso de uso "Generación y Ejecución - Selección"

Figura 4.5: Diagrama del caso de uso "Resultados"

Figura 4.6: Diagrama del caso de uso "Resultados Estadísticos"

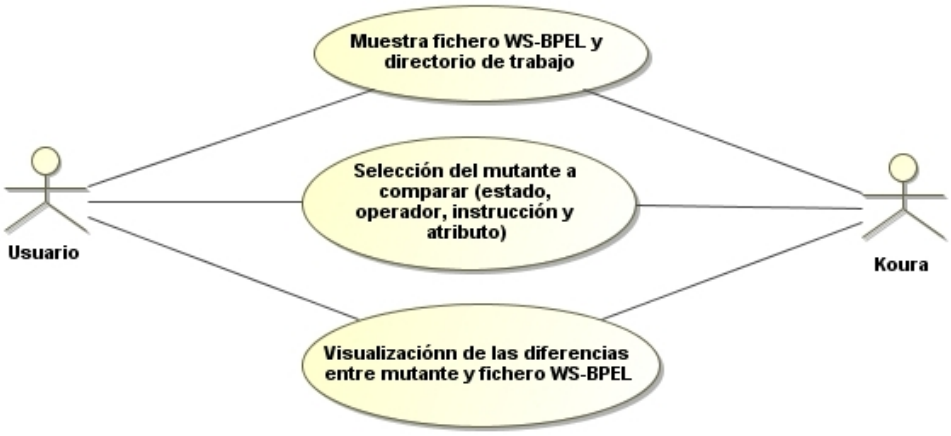

Figura 4.7: Diagrama del caso de uso "Visualización y Comparación"

4.2.4. Especificación de Casos de Uso

Esta tarea es obligatoria en el caso de orientación a objetos y opcional en el análisis estructurado, como apoyo a la obtención de requisitos.

El objetivo de esta tarea es especificar cada caso de uso identificado en la tarea anterior, desarrollando el escenario.

Para completar los casos de uso, es preciso especificar información relativa a:

- Descripción del escenario, es decir, cómo un actor interactúa con el sistema y cual es la respuesta obtenida.
- Precondiciones y poscondiciones.
- Resumen.
- Escenario principal.
- Condiciones de fallo que afectan al escenario, así como la respuesta del sistema (escenarios secundarios).

Caso de uso Analisis

Caso de uso: Análisis

Descripción: El sistema muestra la pantalla de inicio de Koura y el usuario interactúa con la aplicación con la intención de obtener el "Análisis" de un caso de prueba WS-BPEL.

Precondiciones: Ninguna.

Postcondiciones: Se mostrará en pantalla el resultado del "Análisis", los operadores que intervienen en el estudio. Y tras la elección (por parte del usuario) de los operadores que se desea que continúen en el estudio, se mostrará la siguiente sección de Koura que es la pantalla de "Generación y ejecución".

Resumen: El sistema muestra la pantalla de inicio de Koura y el usuario interactúa con ella para obtener los resultados de "Análisis" de un caso de prueba WS-BPEL.

Escenario principal:

1. El usuario selecciona el fichero WS-BPEL con el que se desea realizar el estudio.
2. El usuario crea o selecciona el directorio de trabajo donde se almacenarán los resultados del estudio.
3. El usuario hace clik en "Aceptar" para que se efectúe el "Análisis".
4. Koura realiza la llamada al algoritmo genético correspondiente al "Análisis".

5. Koura interpreta los resultados y muestra por pantalla los operadores que intervienen en el estudio.
6. El usuario selecciona, de los operadores disponibles, aquellos que desea que intervengan.
7. El usuario hace clik en “Siguiente” para continuar.
8. Koura almacena la opción del usuario. En los pasos siguientes trabajará con ésta.
9. Koura muestra la siguiente pantalla de “Generación y Ejecución”.

Escenarios alternativos:

- *a. El usuario decide abandonar Koura.
- 6a. El usuario decide volver a realizar otro análisis y selecciona otro directorio de trabajo y otro fichero WS-BPEL.

Caso de uso Generación y ejecución - Todos

Caso de uso: Generación y ejecución - Todos

Descripción: El sistema muestra la pantalla de "Generación y ejecución" de Koura y el usuario interactúa con ella con la intención de obtener los resultados de la "Generación y ejecución" de un caso de prueba WS-BPEL.

Precondiciones: El caso de prueba WS-BPEL que se estudia, debe tener realizado el proceso de análisis.

Postcondiciones: Según la opción escogida por el usuario en Koura, se obtendrán los resultados de la generación y ejecución del caso de prueba WS-BPEL en el directorio de trabajo correspondiente y se mostrará la siguiente sección de Koura "Resultados".

Resumen: Tras realizar el análisis en la sección anterior de Koura, el usuario quiere realizar el proceso de "Generación y ejecución". Se le ofrecen dos opciones para esto, según la que escoja se le darán unos u otros resultados, pero de igual modo, Koura mostrará posteriormente su pantalla de "Resultados".

Escenario principal:

1. Koura muestra el directorio de trabajo y fichero WS-BPEL con el que se realizó el análisis.
2. El usuario selecciona el fichero de test con el que se va a realizar el proceso de "Generación y ejecución".
3. Koura muestra calcula el número de casos de test y lo muestra por pantalla.
4. El usuario decide realizar el estudio con todos los operadores y hace clik en "Todos".
5. Koura comprueba que se realizó el "Análisis" con anterioridad.
6. Koura realiza la llamada al algoritmo genético para que haga la "Generación y ejecución" con los operadores indicados, almacena los resultados en el directorio de trabajo correspondiente y muestra la siguiente pantalla "Resultados".

Escenarios alternativos:

- *a. El usuario decide abandonar Koura.
- 1a. El usuario quiere realizar la "Generación y ejecución" con otro fichero WS-BPEL y/u otro directorio de trabajo.
- 4a. El usuario decide realizar el estudio con una sección de operadores.
- 5a. Koura detecta que no se realizó el "Análisis" previamente.

- 5a.1 Koura muestra un mensaje de advertencia al usuario de Koura indicándole que no está hecho el "Análisis".
- 5a.2 La pantalla que se muestra de Koura es ahora la pantalla de "Análisis" con el fichero WS-BPEL y el directorio de trabajo que aparecían en la pantalla de "Generación y ejecución".

Caso de uso: Generación y ejecución - Selección

Caso de uso: Generación y ejecución - Selección

Descripción: Seguimos en la pantalla de "Generación y ejecución" de Koura y el usuario decide trabajar con una selección de los operadores.

Precondiciones: Están seleccionados el directorio de trabajo, el fichero WS-BPEL y el fichero de test (lo que ya nos ha dado el número de casos de test) con los que queremos realizar el estudio.

Postcondiciones: Se realizará la llamada al algoritmo genético para que realice con una selección de operadores la "Generación y ejecución".

Resumen: Tras seleccionar el directorio de trabajo, el fichero WS-BPEL y el fichero de prueba (obtenemos el número de casos de test) con los que queremos realizar el estudio, se llamará al algoritmo genético y nos dará los resultados correspondientes a la opción seleccionada.

Escenario principal:

1. Koura muestra el directorio de trabajo y fichero WS-BPEL con el que se realizó el análisis.
2. El usuario selecciona el fichero de test con el que se va a realizar el proceso de "Generación y ejecución".
3. Koura muestra calcula el número de casos de test y lo muestra por pantalla.
4. El usuario introduce en Koura los parámetros con los que va a realizar la llamada.
5. El usuario introduce el valor que quiere que obtenga la semilla de la llamada.
6. El usuario hace clik en "Selección" para que se realice la llamada.
7. Koura comprueba que se realizó el "Análisis" con anterioridad.
8. Koura llama al algoritmo genético para que se realice las operaciones de "Generación y ejecución" según la opción seleccionada por el usuario "Selección".

Escenarios alternativos:

- *a. El usuario decide abandonar Koura.
- 1a. El usuario quiere realizar la "Generación y ejecución" con otro fichero WS-BPEL y/u otro directorio de trabajo.
- 4a. El usuario selecciona un fichero gamera.conf para cargar la configuración.
  - 4a.1 Koura lee el fichero gamera.conf seleccionado y muestra los valores por pantalla.
  - 4a.2 El usuario si lo desea puede modificar alguno de los valores de la configuración.
- 7a. Koura detecta que no se realizó el "Análisis" previamente.
  - 7a.1 Koura muestra un mensaje de advertencia al usuario de Koura indicándole que no está hecho el "Análisis".
  - 7a.2 La pantalla que se muestra de Koura es ahora la pantalla de "Análisis" con el fichero WS-BPEL y el directorio de trabajo que aparecían en la pantalla de "Generación y ejecución".

Caso de uso Resultados

Caso de uso: Resultados

Descripción: El usuario viene de realizar su estudio de un caso de prueba WS-BPEL y quiere visualizar los resultados. Puede escoger si quiere visualizar los resultados estadísticos o bien quiere realizar la comparación de mutantes.

Precondiciones: Se ha realizado los procesos de "Análisis + Generación + Ejecución" en el caso de prueba WS-BPEL

Postcondiciones: Según si el usuario ha escogido la opción de visualizar los resultados estadísticos o la comparativa de mutantes, Koura mostrará la pantalla correspondiente y mostrará los resultados tras interpretar los ficheros resultantes de las operaciones.

Resumen: Se ha realizado los procesos de "Análisis + Generación + Ejecución" en el caso de prueba WS-BPEL y según el usuario escoja la opción de ver los resultados estadísticos o la comparativa de mutantes, Koura activará la pantalla correspondiente y mostrará los resultados tras interpretar los ficheros.

Escenario principal:

1. Koura muestra la pestaña para la visualización de resultados estadísticos, con el fichero WS-BPEL y el directorio de trabajo con el que se ha venido trabajando.

2. El usuario opta por visualizar los resultados estadísticos y hace clik en "Calcular".

3. Koura comprueba si se han realizado los procesos de "Análisis + Generación + Ejecución".

4. Koura interpreta los ficheros resultantes de los procesos anteriores y realiza unos cálculos y los visualiza junto con otros valores correspondientes a los ficheros resultados.

Escenarios alternativos:

- *a. El usuario decide abandonar Koura.

- 1a. El usuario decide cambiar el fichero WS-BPEL y/o el directorio de trabajo.

- 2a. El usuario decide visualizar los resultados mediante la comparativa de mutantes. Y le da a la pestaña de "Ver mutantes".

- 3a. Koura comprueba que no se realizaron los procesos de "Generación + Ejecución", muestra una ventana de aviso al usuario indicándoselo y lleva al usuario a la pestaña de "Generación y Ejecución".

Caso de uso: Ver mutantes

Descripción: Se han terminado de realizar los procesos de "Análisis + Generación + Ejecución" y el usuario desea visualizar las comparativas de los mutantes.

Precondiciones: El usuario viene de la pestaña "Resultados", así que las precondiciones son las mismas que las de esta pestaña.

Postcondiciones: Se mostrarán en pantalla la comparativa del fichero original y el fichero mutante que seleccione el usuario.

Resumen: El usuario selecciona la opción de visualizar los resultados mediante la comparativa de los mutantes. Tras seleccionar el mutante que se desea comparar se visualizarán las diferencias entre éste y el original.

Escenario principal:

1. Koura muestra la pestaña de "Ver Mutantes" con el fichero WS-BPEL y el directorio de trabajo con el que estaba realizando el estudio.
2. El usuario escoge el estado en el que se encuentra el mutante que desea comparar.
3. Koura, según el estado seleccionado por el usuario, visualiza los operadores que están en dicho estado.
4. El usuario selecciona de los operadores mostrados por Koura aquél que desea comparar.
5. Koura, según el operador que ha seleccionado el usuario, visualiza los valores correspondientes a la instrucción y atributo **I-A** del operador asociado según el estado del operador.
6. El usuario selecciona uno de los valores **I-A** que Koura ha establecido que cumple con los prerequisitos escogidos por el usuario con anterioridad.
7. El usuario hace clik en "Visualizar".

Escenarios alternativos:

- *a. El usuario decide abandonar Koura.
- 1a. El usuario decide cambiar el fichero WS-BPEL y/o el directorio de trabajo.
- 3a. Koura detecta que según el estado que ha seleccionado el usuario, no existe ningún operador.
  - 3a.1 Muestra en pantalla el listado en blanco.
- 3b. Koura detecta que el estado que ha seleccionado el usuario es "Estado".
  - 3b.1 Koura limpia los listados de los operadores, y de los detalles correspondientes a los valores **I-A**. También limpia las áreas de texto donde se visualizan los códigos.

4.2.5. Modelo conceptual de datos

El objetivo de esta frase es la identificación de las necesidades de información de cada uno de los procesos que conforman el sistema de información con el fin de obtener un modelo de datos que contemple todas las entidades, relaciones y atributos necesarios para dar respuesta a dichas necesidades.

Se ha usado el modelo de datos de: Diagrama de clases. Este modelado incluye: clases, asociaciones, atributos, restricciones de seguridad... Se trata de uno de los modelos más comunes en el modelado de sistemas orientados a objetos. Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones y sus relaciones entre ellos.

Las clases representan los bloques de construcción más importantes de cualquier sistema orientado a objetos. Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

Esta notación es independiente de cualquier lenguaje de programación. El primer bloque representa el nombre de la clase, el segundo bloque contiene los atributos y el tercer bloque contiene las operaciones.

No es necesario mostrar todas las características. A veces las clases tienen tantas características, que no es conveniente mostrarlas todas. En estos casos también se pueden organizar las características usando estereotipos.

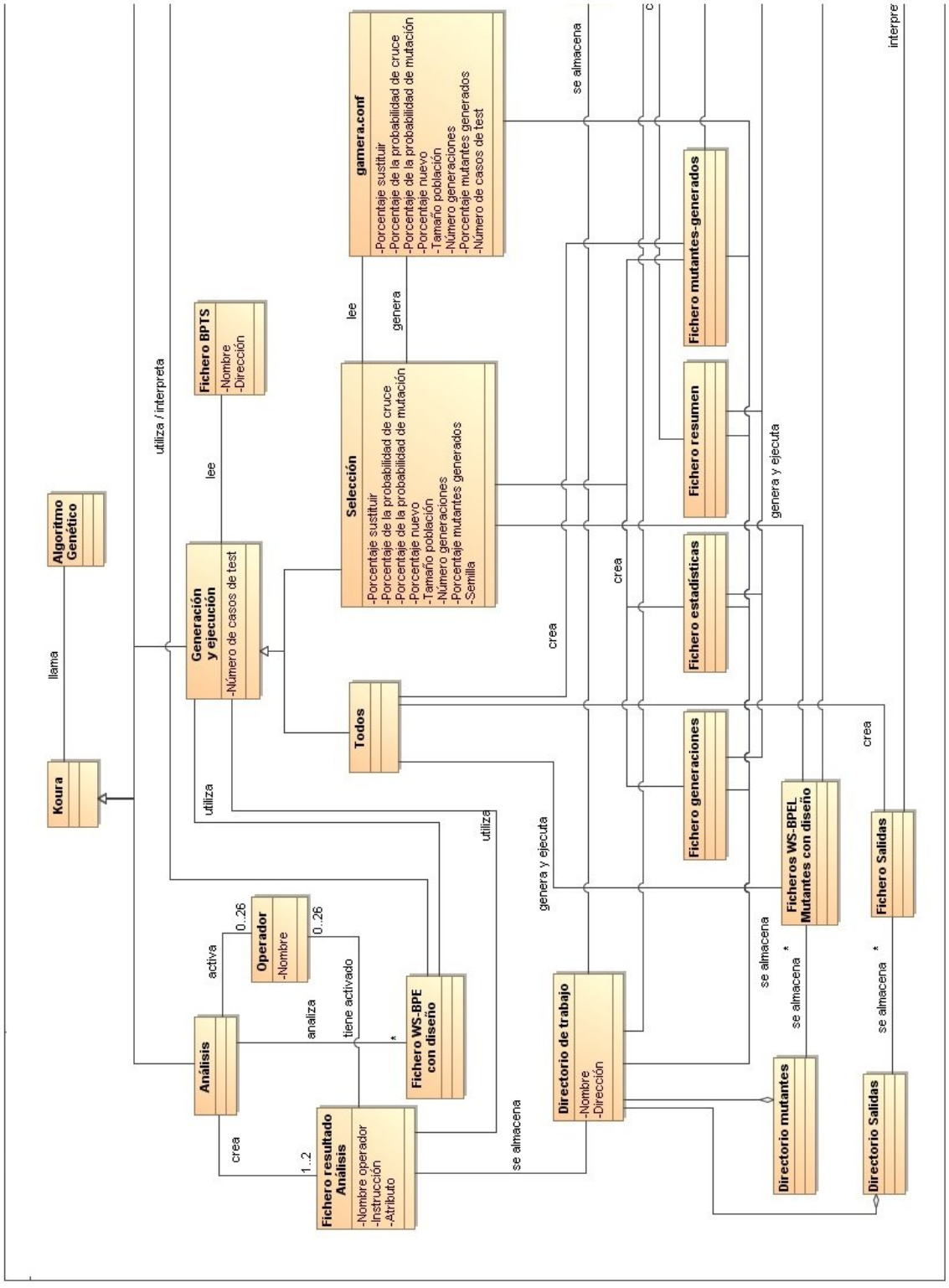

Figura 4.8: Modelo Conceptual de Datos - Primera parte

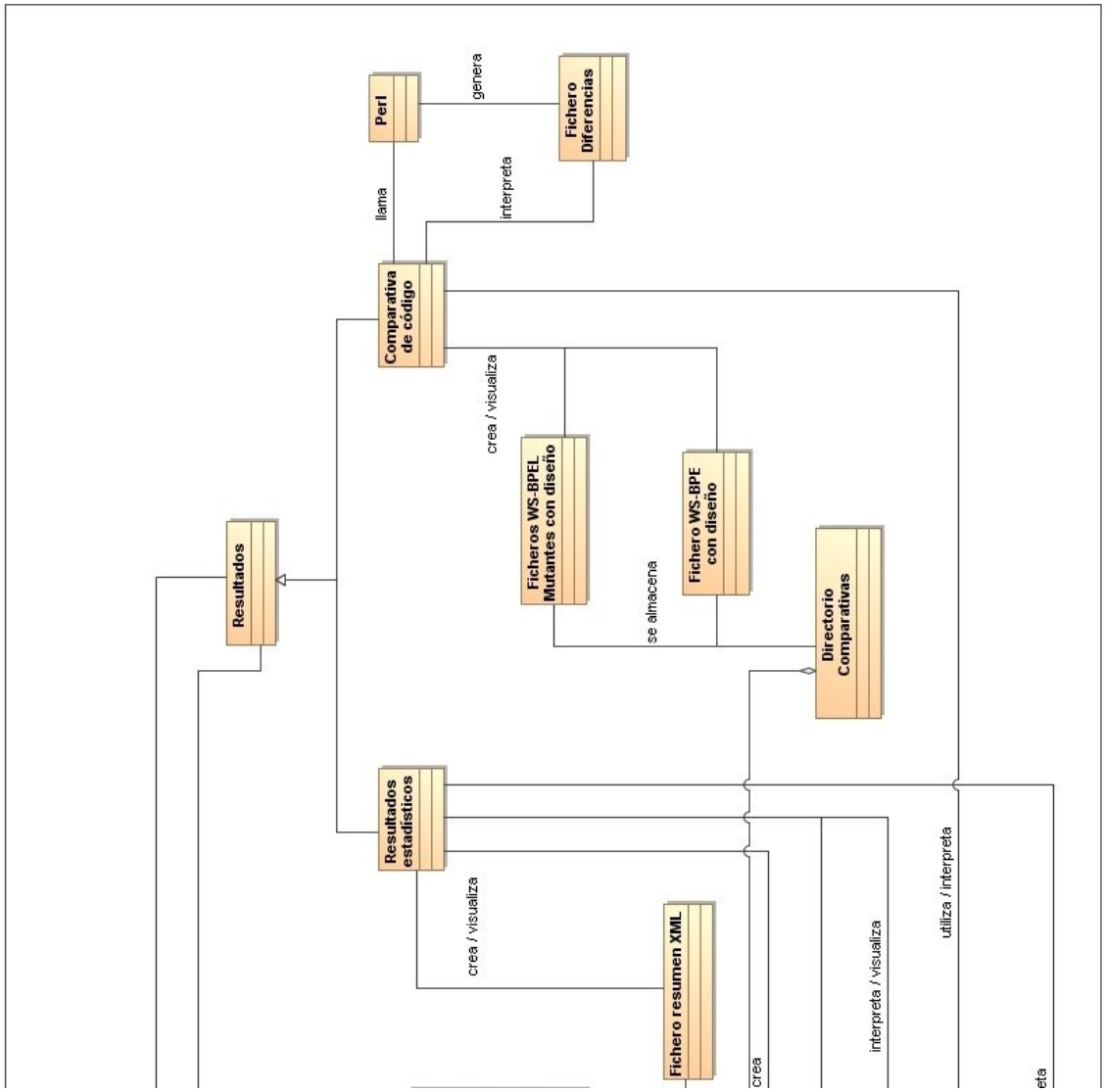

Figura 4.9: Modelo Conceptual de Datos - Segunda parte

4.2.6. Modelo de comportamiento del sistema

El modelado del comportamiento del sistema nos permite obtener la especificación del comportamiento. Las técnicas que emplearemos para el modelado serán los diagramas de secuencia y los contratos de las operaciones.

Los diagramas de secuencia muestran la secuencia de eventos que producen los actores del sistema. Nos ayudan a la identificación de las operaciones del sistema. Los contratos de las operaciones nos permiten describir el efecto de las operaciones del sistema.

4.2.7. Diagramas de secuencia de sistemas y Contrato de las operaciones del sistema

Caso de uso: Inicio

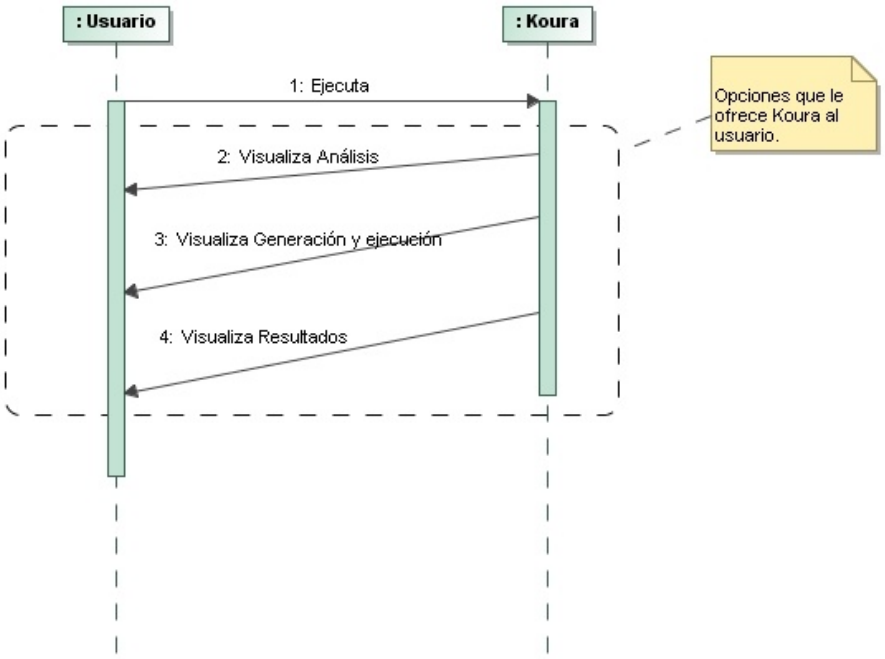

Figura 4.10: Diagrama de comportamiento - Inicio

Contratos de las operaciones

Operación: Ejecuta (mono Mutantes.exe)
Responsabilidades: Inicia Koura.
Precondiciones: Ninguna.
Postcondiciones: Se ejecuta la aplicación Koura.

Operación: Visualiza Analisis
Responsabilidades: Visualizar la pantalla de Análisis.
Precondiciones: El usuario hace clik en la pestaña de Análisis. Al inicio de Koura se visualiza por defecto.
Postcondiciones: Se visualiza la pantalla de Análisis.

Operación: Visualiza Generación y ejecución
Responsabilidades: Visualizar la pantalla de Generación y ejecución.
Precondiciones: El usuario hace clik en la pestaña de Generación y ejecución.
Postcondiciones: Se visualiza la pantalla de Generación y ejecución.

Operación: Visualiza Resultados
Responsabilidades: Visualizar la pantalla de resultados que seleccione el usuario.
Precondiciones: El usuario hace clik en la pestaña de Resultados que desea.
Postcondiciones: Se visualiza la pantalla de Resultados que el usuario haya seleccionado.

Caso de uso: Análisis

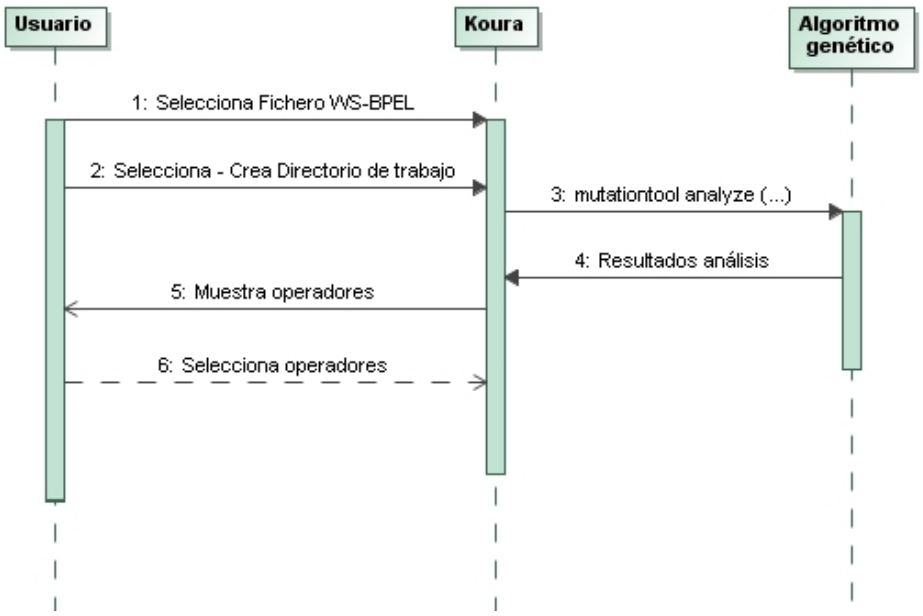

Figura 4.11: Diagrama de comportamiento - Análisis

Contratos de las operaciones

Operación: Selecciona
Responsabilidades: Escoger el fichero WS-BPEL o el directorio de trabajo que se va a emplear para realizar el estudio.
Precondiciones: Que el fichero WS-BPEL o el directorio de trabajo existan.
Postcondiciones: El fichero WS-BPEL o el directorio de trabajo queda seleccionado.

Operación: Crea
Responsabilidades: Se crea en el sistema el directorio de trabajo que ha denominado el usuario.
Precondiciones: No tiene que existir el directorio de trabajo que desea crear el usuario en la ubicación señalada.
Postcondiciones: Se crea el directorio de trabajo seleccionado por el usuario en la ubicación escogida.

Operación: mutationtool analyze(Fichero WS-BPEL, Directorio de trabajo)
Responsabilidades: Koura llama al algoritmo genético con la operación anterior para que se produzca el análisis sobre el fichero WS-BPEL seleccionado.

Precondiciones: Tienen que haberse escogido el fichero WS-BPEL y el directorio de trabajo.
Postcondiciones: Se realizará el análisis del fichero WS-BPEL seleccionado y el **fichero resultado Análisis** que **Genera** se almacenarán en el directorio de trabajo.

Operación: Resultados análisis
Responsabilidades: Los ficheros resultados del análisis se ubican en el directorio de trabajo para su posterior uso.
Precondiciones: Los ficheros se ha generado.
Postcondiciones: Los ficheros resultados del análisis se almacenan en el directorio de trabajo.

Operación: Muestra operadores
Responsabilidades: Koura lee el **fichero resultado Análisis** y según los operadores que tenga activados (en la columna de la instrucción, segunda columna, aparece un número distinto de cero) los marcará con un tick en la pantalla de Análisis.
Precondiciones: Se ha realizado el análisis del fichero WS-BPEL, por lo que existe el fichero resultado Análisis.
Postcondiciones: En la pantalla de Análisis de Koura, aparecerán habilitados aquellos operadores que están activos después del análisis.

Operación: Selecciona operadores
Responsabilidades: El usuario selecciona, si lo desea, un conjunto de los operadores que Koura le indica que están activos.
Precondiciones: Koura ha visualizado previamente aquellos operadores que están activos, o mejor dicho que intervienen, en el estudio del fichero WS-BPEL que ha sido seleccionado.
Postcondiciones: El usuario selecciona de la pantalla de Análisis aquellos operadores activos que le interesen.

Caso de uso: Generación y ejecución - Todos

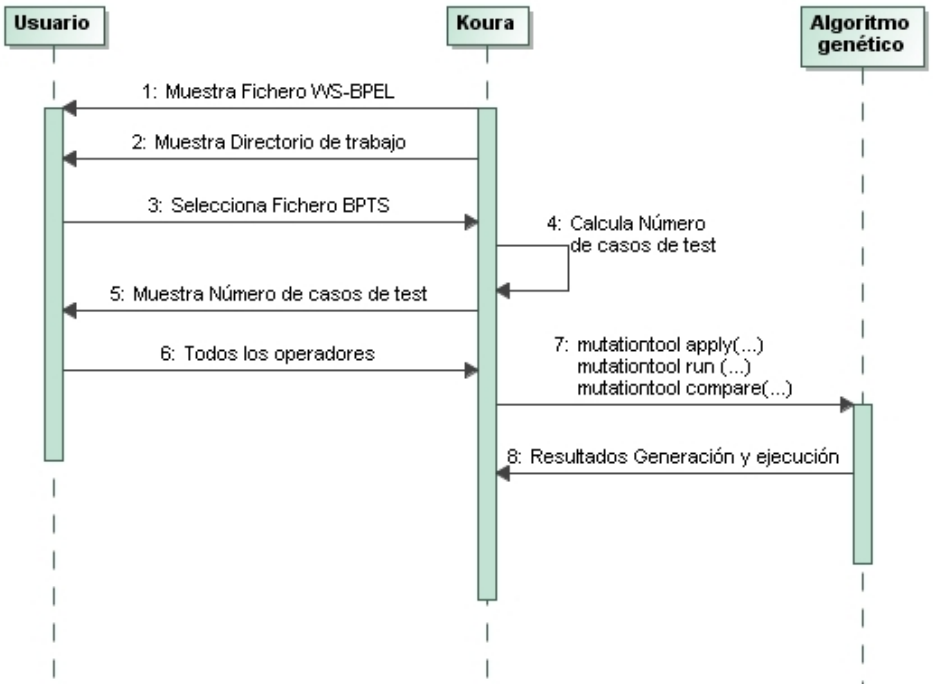

Figura 4.12: Diagrama de comportamiento - Generación y ejecución: Todos

Contratos de las operaciones

Operación: Muestra Fichero WS-BPEL / Directorio de trabajo Responsabilidades: Koura lee de la sección anterior el fichero WS-BPEL y el directorio de trabajo que se seleccionó para el estudio y lo muestra en esta nueva sección.
Precondiciones: Se ha realizado un análisis previamente, con lo que se ha escogido el fichero WS-BPEL y el directorio de trabajo.
Postcondiciones: Se muestran en pantalla el fichero WS-BPEL y el directorio de trabajo.

Operación: Selecciona fichero BPTS
Responsabilidades: El usuario selecciona el fichero de test (BPTS) para calcular el Número de casos de test que se emplearán en el estudio.
Precondiciones: Ninguna.
Postcondiciones: El fichero de test queda seleccionado.

Operación: Calcula Número de casos de test
Responsabilidades: Koura interpreta el fichero BPTS contando el número de etiquetas

de cierre BPTS para calcular el número de casos de test que se pueden realizar y visualiza el número por pantalla.
Precondiciones: Se ha seleccionado un fichero de test.
Postcondiciones: Se interpreta el fichero de test y se visualiza el número de casos de test que se pueden realizar en la pantalla Generación y ejecución.

Operación: Muestra Número de casos de test
Responsabilidades: Visualizar en pantalla tras realizar los cálculos el número de casos de test que se van a realizar en el estudio de la generación y ejecución.
Precondiciones: Se ha seleccionado un fichero de test BPTS y se han realizado los cálculos.
Postcondiciones: Aparece por pantalla el número de casos de test para el estudio.

Operación: Todos los operadores
Responsabilidades: Recopilar todos los valores y datos necesarios para realizar la llamada al algoritmo genético indicándole que se va a realizar la generación y ejecución con todos los operadores.
Precondiciones: Todos los datos necesarios para la generación y ejecución de todos los operadores están introducidos.
Postcondiciones: Koura realiza la llamada al algoritmo genético para el estudio con todos los operadores.

Operación: mutationtool apply(Fichero WS-BPEL, OP, INS, ATR), mutationtool run(Fichero BPTS, Fichero WS-BPEL), mutationtool compare(Fichero BPTS, Fichero WS-BPEL, Fichero.out, FicheroMut)
Responsabilidades: Koura llama al algoritmo genético con estas instrucciones para realizar la generación y ejecución de todos los operadores seleccionados por el usuario en la pantalla de Análisis de Koura.
Precondiciones: El fichero WS-BPEL, el fichero BPTS y el directorio de trabajo han sido seleccionados y se ha realizado un análisis previo del fichero WS-BPEL que está almacenado el el directorio de trabajo seleccionado.
Postcondiciones:

- Si no se ha seleccionado un fichero WS-BPEL al que se le haya realizado un análisis previo que esté almacenado en el directorio de trabajo, nos dará Koura un aviso y nos llevará a la pantalla de Análisis para que se lo hagamos.
- Si no se ha seleccionado un fichero BPTS, con el que obtenemos el número de casos de test, nos avisará Koura con un mensaje de advertencia.
- En caso de que esté todo correcto, con la primera instrucción se genera para cada mutante seleccionado, OPERADOR, INSTRUCCION y ATRIBUTO un ficheromutante.bpel. Estos ficheros **se almacenan** dentro del directorio de trabajo en una carpeta llamada “mutantes”. Con la siguiente obtenemos un fichero fichero.out

que nos servirá para la siguiente instrucción, en la que para cada mutante que se nos haya generado nos dará un fichero salida que nos indicará mediante los valores 0, 1 y 2 si el test que viene denominado según su posición en el fila que contiene el fichero lo ha matado, ha dado error o bien sigue vivo. Estos ficheros **se almacenan** en nuestro directorio de trabajo en una carpeta denominada "Salida". Los directorios "mutantes" y "Salida" **se ubican** dentro del directorio de trabajo.

Operacion: Resultados Generación y ejecución

Responsabilidades: El algoritmo genético crea para la generación y ejecución el fichero mutantes-generados y los ficheros de salida. Los ficheros de salida se generan tantos como OPERADOR * INSTRUCCIÓN * ATRIBUTO y contienen una fila con los valores 0, 1 o 2, con los que se nos indica si el mutante sigue vivo 0, muerto 1 o ha causado error 2. en el caso de estar muerto, el 1 estará en la columna que está en la posición correspondiente al test que lo ha matado. El algoritmo genético genera y ejecuta los ficheros mutantes, son tantos como OPERADOR * INSTRUCCIÓN * ATRIBUTO. Estos ficheros contienen una diferencia respecto al fichero original WS-BPEL.
Precondiciones: Koura ha realizado la llamada para que el algoritmo genético realice la generación y ejecución con todos los operadores.
Postcondiciones: Se crea el fichero mutantes-generados que **se almacena** en el directorio de trabajo, y los ficheros de salida, que **se almacenan** en el directorio "Salida", que **se ubica** en el directorio de trabajo. Se generan y ejecutan los ficheros mutantes que **se almacenan** en el directorio "mutantes", el cual **se ubica** en el directorio de trabajo.

Caso de uso: Generación y ejecución - Selección

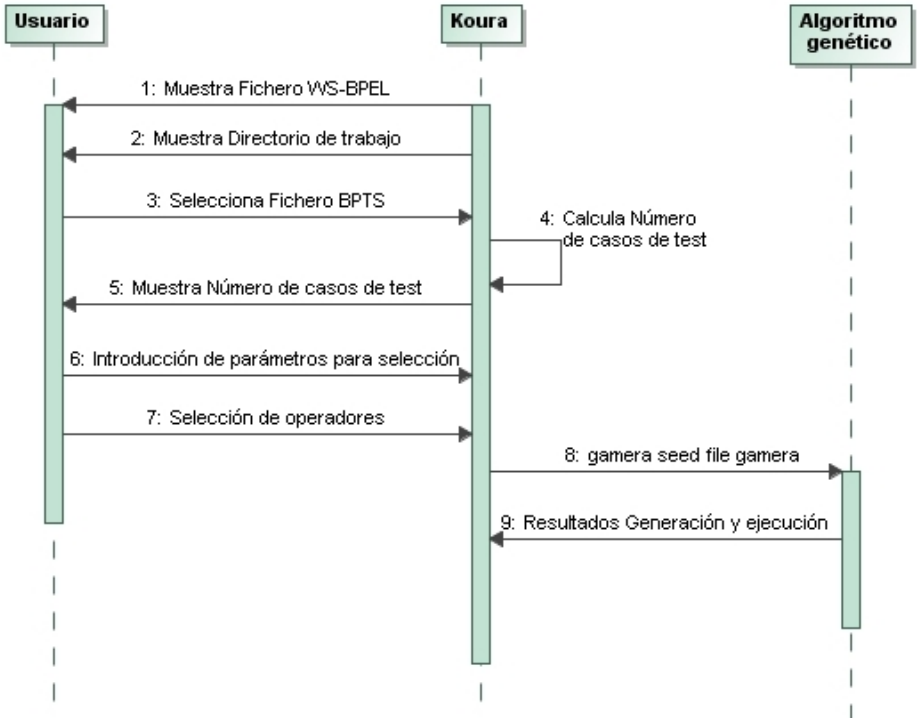

Figura 4.13: Diagrama de comportamiento - Generación y ejecución: Selección

Contratos de las operaciones

Operación: Introducción de parámetros para selección
Responsabilidad: El usuario quiere introducir los valores de los parámetros para la generación y ejecución de una selección de mutantes. Se le presentan las siguientes opciones:

1. Introducir uno a uno los valores de los diferentes parámetros necesarios para el fichero gamera.conf que se empleará en la "Generación y ejecución" de una selección de mutantes.

2. Seleccionar un fichero gamera.conf para cargar una configuración que ya se realizó en otro estudio.

3. Modificar algún valor de los que se cargaron de la configuración previa de un fichero gamera.conf, se modificará directamente en la pantalla "Generación y ejecución".

Precondiciones: En la última de las opciones, que en la pantalla aparezcan los valores de los diferentes parámetros que se han configurado desde un fichero gamera.conf.
Postcondiciones: Aparecen en pantalla de "Generación y ejecución" de Koura los valores de los parámetros introducidos, de una forma u otra, por el usuario.

Operación: Selección de operadores
Responsabilidades: Recopilar todos los valores y datos necesarios para realizar la llamada al algoritmo genético indicándole que se va a realizar la generación y ejecución con selección de los operadores.
Precondiciones: Todos los datos necesarios para la generación y ejecución de una selección de los operadores están introducidos.
Postcondiciones: Koura realiza la llamada al algoritmo genético para el estudio con una selección de los operadores.

Operación: gamera(seed file gamera.conf)
Responsabilidad: Koura llama al algoritmo genético para que realice la "Generación y ejecución" de la selección de mutantes según los parámetros que el fichero gamera.conf le indique y según la semilla que se haya introducido en la pantalla de "Generación y ejecución".
Precondiciones: Se han introducido todos los parámetros necesarios para la creación del fichero gamera.conf.
Postcondiciones: Se realiza la "Generación y ejecución" de la selección de los mutantes que le indiquen los parámetros del fichero gamera.conf.

Operación: Resultados Generación y ejecución
Responsabilidad: El algoritmo genético crea para la generación y ejecución el fichero mutantes-generados, el fichero generaciones y el fichero estadísticas. También se generan y ejecutan los ficheros mutantes, son tantos como se indican en el fichero gamera.conf. Estos ficheros contienen una diferencia respecto al fichero original WS-BPEL.
Precondiciones: El usuario ha seleccionado los ficheros necesarios para la Generación y ejecución.
Postcondiciones: Se crea el fichero mutantes-generados, el fichero generaciones y el fichero estadísticas que **se almacenan** en el directorio de trabajo. Se generan y ejecutan los ficheros mutantes que **se almacenan** en el directorio "mutantes", el cual **se ubica** en el directorio de trabajo.

Caso de uso: Resultados

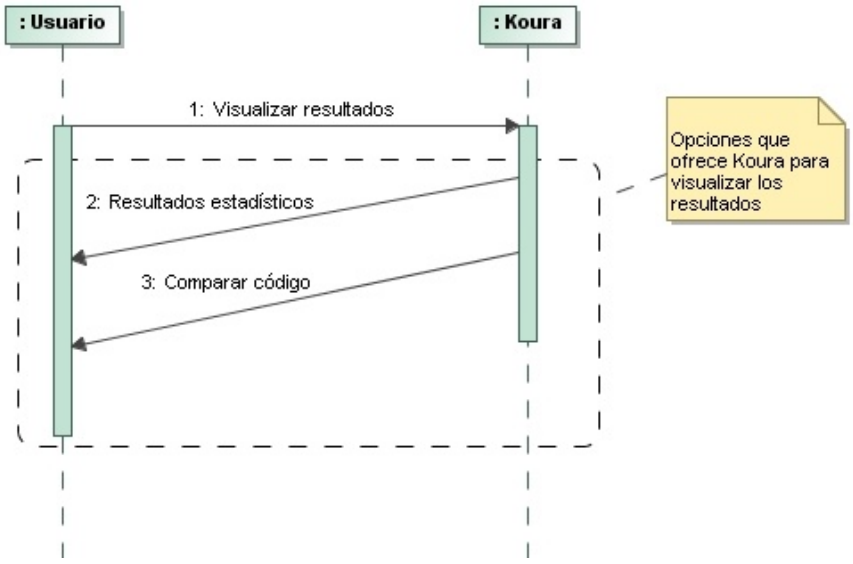

Figura 4.14: Diagrama de comportamiento - Resultados

Contratos de las operaciones

Operación: Visualizar resultados
Responsabilidad: El usuario ha realizado el estudio de los mutantes y quiere visualizar los resultados.
Precondiciones: Se ha realizado el estudio de los mutantes (Análisis + Generación + Ejecución).
Postcondiciones: Koura presenta dos opciones para ver los resultados del estudio; mostrando unos resultados estadísticos y mediante la comparación de código.

Operación: Resultados estadísticos
Responsabilidad: Koura muestra al usuario la pantalla de "Resultados".
Precondiciones: El usuario ha deseado visualizar los resultados del estudio mutante que ha realizado previamente mediante unos valores estadísticos.
Postcondiciones: koura muestra la pantalla donde se visualizan los resultados estadísticos de los estudios.

Operación: Comparar código
Responsabilidad: Koura muestra al usuario la pantalla de "Ver mutantes".
Precondiciones: El usuario ha deseado visualizar los resultados del estudio mutante que

ha realizado previamente mediante la comparación de código.
Postcondiciones: Koura muestra la pantalla donde se visualizan la comparación de código de los estudios.

Caso de uso: Resultados - Resultados estadísticos

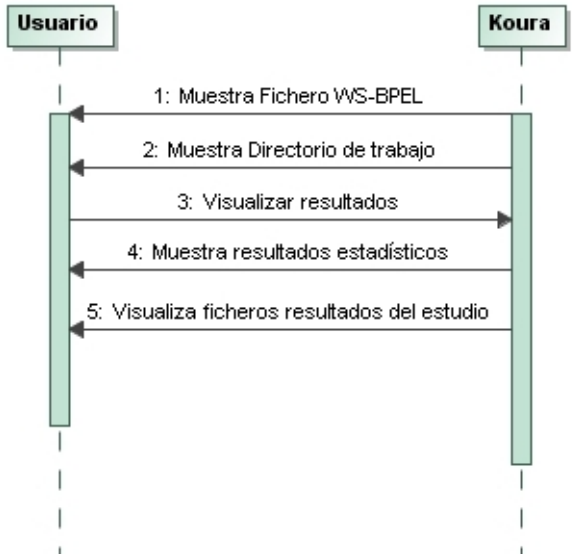

Figura 4.15: Diagrama de comportamiento - Resultados: Resultados estadísticos

Contratos de las operaciones

Operación: Visualizar Resultados
Responsabilidades: Koura muestra los resultados estadísticos que se han generado con el fichero WS-BPEL en el estudio almacenado en el directorio de trabajo que se seleccionaron en el análisis. Para ello comprueba que se ha realizado la generación y ejecución de los mutantes, y lee el fichero mutantes-generados para realizar unos cálculos que servirán para generar los resultados que el usuario desea visualizar.
Precondiciones: Se ha tenido que realizar previamente el estudio del fichero WS-BPEL seleccionado y éste tiene que estar almacenado en el directorio de trabajo que escogido.
Postcondicones:

- Si no se ha realizado la generación y ejecución de los mutantes, nos envía a la pantalla de Koura "Generación y ejecución", tras avisarnos mediante un mensaje de advertencia.
- Si se ha realizado la generación y ejecución, Koura realiza unos cálculos sobre los resultados en los que está interesado el usuario.

Operación: Visualiza los ficheros resultados del estudio
Responsabilidades: Koura genera un fichero resumen, tanto en texto plano como en lenguaje xml, que visualiza por pantalla los valores estadísticos, (ambos se visualizan,

pero de diferente modo), que obtuvo de los cálculos que realizó tras interpretar el fichero mutantes-generados. Ambos **se almacenan** en el directorio de trabajo. Para el resto de los ficheros (mutantes-generados, generaciones, estadísticas y resumen), Koura hace una llamada al editor "emacs" para que se visualicen los ficheros cuando el usuario desee acceder a ellos. Dependiendo de si se hizo un estudio con todos o con una selección de mutantes, se podrán visualizar unos u otros.
Precondiciones: Se ha realizado un estudio mutante del caso de prueba y para la generación del fichero resumen, que se hayan realizado los cálculos del fichero mutantes-generados.
Postcondiciones: Se visualizan en pantalla "Resultados" los ficheros resúmenes que han sido generados en Koura. Para el resto de ficheros, se abre el editor de texto emacs con el fichero resultado al que se desea acceder. En el caso de estar en un estudio que hemos procesado todos los mutantes, se podrán visualizar mutantes-generados y resumen y en el que se procesen un selección de mutantes, se podrán visualizar todos.

Caso de uso: Resultados - Ver mutantes

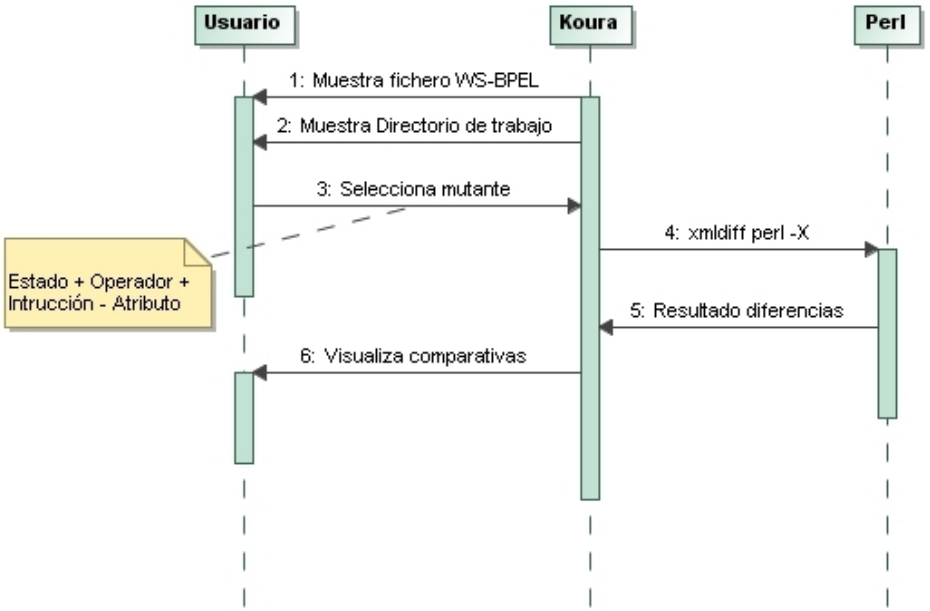

Figura 4.16: Diagrama de comportamiento - Resultados: Ver Mutantes

Contratos de las operaciones

Operación: Selecciona mutante (Estado, Operador, I-A)
Responsabilidades: El usuario selecciona del listado de estados, el estado del mutante que desea visualizar (Vivo, muerto o error). Cuando selecciona uno de ellos, Koura rellena la lista de operadores con aquellos que cumplen el requisito del estado seleccionado. Una vez seleccionado el operador, Koura rellenará la última de las listas que contienen los valores instrucción y atributo (I-A) y el usuario seleccionará qué detalle de estos valores quiere visualizar, concretando así el mutante que desea comparar. Koura también tiene que comprobar que se ha realizado la generación y ejecución del mutante.
Precondiciones: Tiene que haberse realizado un estudio mutante (Análisis + Generación + Ejecución) y el usuario desea obtener los resultados mediante la comparativa de código.
Postcondiciones:

- Koura tras comprobar no se ha realizado la generación y ejecución de mutantes, nos envía a la pantalla "Generación y ejecución" tras mostrarnos un mensaje de advertencia.

- Koura comprueba que se ha realizado la generación y ejecución y ya conoce el mutante que el usuario desea visualizar para compararlo con el fichero BPEL original.

Operación: perl xmldiff -X (Fichero WS-BPEL, Fichero mutante)
Responsabilidades: Koura llama a Perl mediante un script para obtener las diferencias entre el fichero WS-BPEL y el fichero mutante.
Precondiciones: Se ha seleccionado un fichero mutante para la comparación.
Postcondiciones: Se obtienen las diferencias entre los ficheros WS-BPEL y el fichero mutante.

Operación: Resultado diferencias
Responsabilidades: Koura crea un fichero xml denominado diferencias, donde se almacenarán las diferencias que el script de Perl ha encontrado entre los ficheros WS-BPEL y el fichero mutante.
Precondiciones: Se ha llamado al script de Perl que nos dice las diferencias entre el fichero WS-BPEL y el mutante.
Postcondiciones: Se crea el fichero diferencias.xml con las diferencias entre los ficheros, incluyéndose unos caracteres que nos indicarán a qué fichero corresponde cada línea diferente.

Operación: Crea - Visualiza
Responsabilidades: Con el fichero diferencias.xml, Koura crea un fichero mutante con las diferencias encontradas con el script de Perl, que nos permitirá visualizarlo en él con diseño (se subrayan en rojo las líneas dispares). Del mismo modo crea un fichero WS-BPEL con las diferencias encontradas con el script de Perl con diseño (se subrayan en verde las líneas dispares).
Precondiciones: Koura conoce las diferencia entre los ficheros.
Postcondiciones: Se crean los ficheros mutante con diseño y BPEL con diseño y aparecen en la pantalla "Ver mutantes" en Koura, subrayadas las partes que son dispares.

4.2.8. Definición de Interfaces de Usuario

Especificamos las interfaces entre el sistema y el usuario: formatos de pantallas, diálogos e informes, principalmente. El objetivo es realizar un análisis de los procesos del sistema de información en los que se requiere una interacción del usuario, con el fin de crear una interfaz que satisfaga todos los requisitos establecidos, teniendo en cuenta los diferentes perfiles a quiénes va dirigido.

El procedimiento que se ha seguido para el desarrollo de las interfaces de usuario ha sido un proceso en espiral, al igual que con todo el proyecto. En mi caso, el Cliente se ve representado por la gran mayoría de los profesores que forman parte del grupo de investigación SPI&FM.

Como se comentó al principio, este proyecto va a desarrollarse para el grupo de investigación SPI&FM por lo que ellos fueron diciéndome las necesidades de la aplicación a desarrollar y de ahí recogía los requisitos de la aplicación.

Siguiendo los requisitos y la estética de la interfaz de μJava y μEclipse se realizó un primer esbozo de las cuatro pestañas de Koura. Tras mostrar al grupo de investigación este primer paso, se siguió con el modelo en espiral y me volvieron a decir algunas mejoras o necesidades para la aplicación.

Tras varios ciclos espirales llegamos a una primera versión de Koura que cumplía los requisitos de nuestro Cliente. A partir de ahí se siguió investigando con la herramienta monodevelop para mejorar la estética de Koura, del mismo modo se siguieron algunos pasos de la guía de estética [28] que nos ayudaron a mejorar su aspecto.

En las últimas versiones de Koura se recortó su tamaño, en especial se recortó en vertical para crear una versión de Koura para portátiles pequeños, en la que conservan los mismos componentes y añadimos "scrollbars"en los laterales para que conserve la distancia entre ellos.

Detalles de las mejoras a comentar

Para que la visión de Koura fuese más estilizada y ergonómica, se destacan los siguientes cambios:

- La selección del "Directorio de trabajo" está diseñada con el mismo componente que la selección del Fichero WS-BPEL. Antes el "Directorio de trabajo" se escribía por pantalla y se almacenaba dentro del directorio del caso de estudio del fichero WS-BPEL seleccionado.

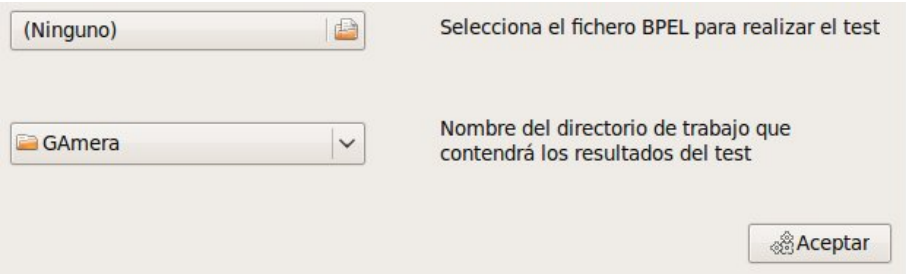

Figura 4.17: Mejora I

- Los checkbox al principio no contaban con una opción que les dejase "inhabilitados" para que el usuario supiera que el correspondiente operador al que representaba, no podía emplearse. De ahí a que se le aplicara un diseño con el que se observaba que no era aplicable dicho operador. Finalmente con la nueva versión de monodevelop, se ha incorporado una nueva opción que nos permite "inhabilitar" el checkbox, consiguiendo nuestro principal objetivo.
- El componente de pestañas que contiene los checkbox se ha dispuesto de esa forma para la división de los diferentes tipos de operadores existentes. Para que estéticamente se viera como un componente importante dentro de la pestaña, se le puso el nombre de cada una de estas pestañas de manera horizontal en el lado izquierdo.

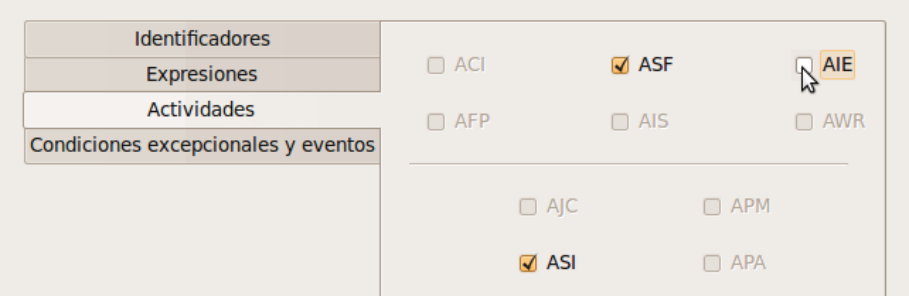

Figura 4.18: Mejoras II y III

- A la hora de introducir los valores de "Opciones Avanzadas" en Koura, al principio eran introducidos a mano, ahora también se nos ofrece la opción de cargar la configuración desde un fichero "gamera.conf", sin que esté ligado al estudio, solamente nos servirá para asignar los valores a los parámetros. Añadimos del mismo modo un componente con el que podemos incrementar o decrementar en una unidad el valor.

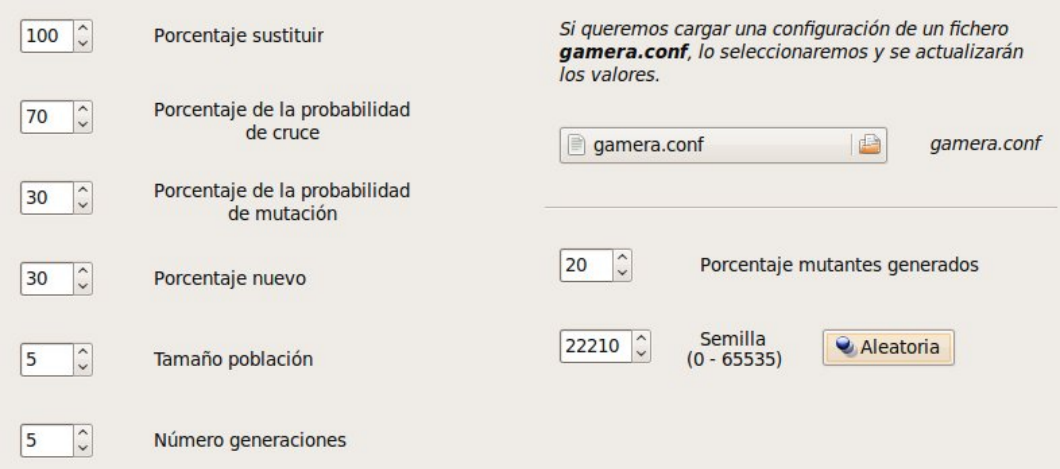

Figura 4.19: Mejora IV

- Cuando Koura detecta que nos falta un fichero o que no hemos realizado un paso, nos mostrará un aviso que nos indicará qué debemos hacer.

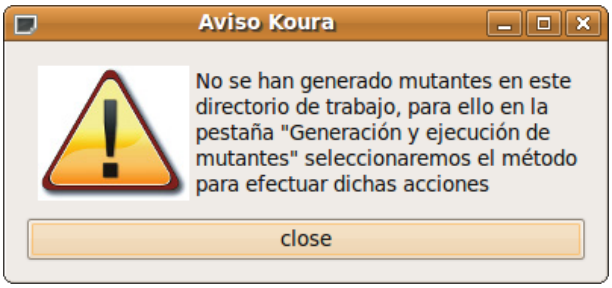

Figura 4.20: Mejora V

- A la hora de visualizar el resumen estadístico de los resultados del estudio, se amplió el área del resumen ya que se consideró la parte más importante de la pestaña, del mismo modo se le dio estilo a los títulos para que ofrecieran un entorno más llamativo.
- Para no confundir al usuario, los botones para la visualización de los ficheros resultados que aparecen al final de la pestaña, se activan según el estudio realizado, evitando así que el usuario pueda confundirse y pueda provocar un error en Koura.

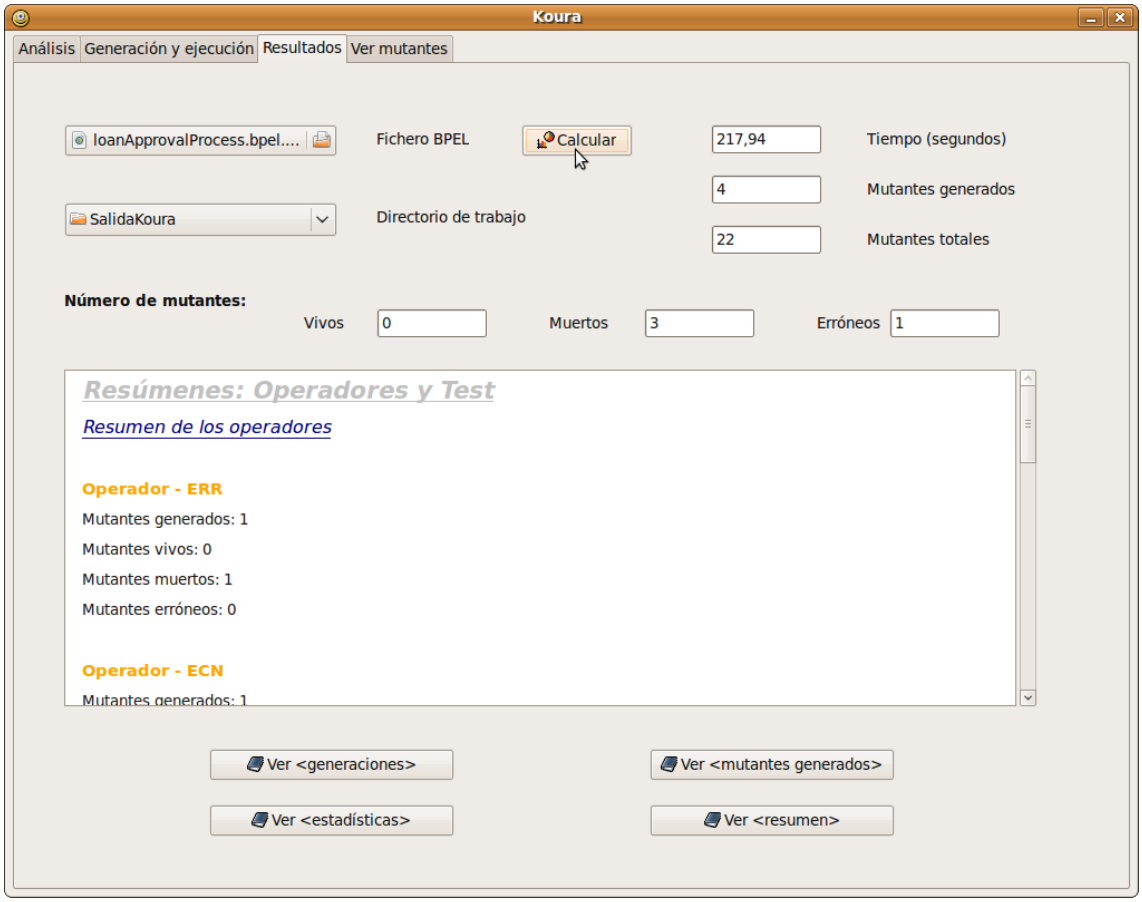

Figura 4.21: Mejora VI y VII

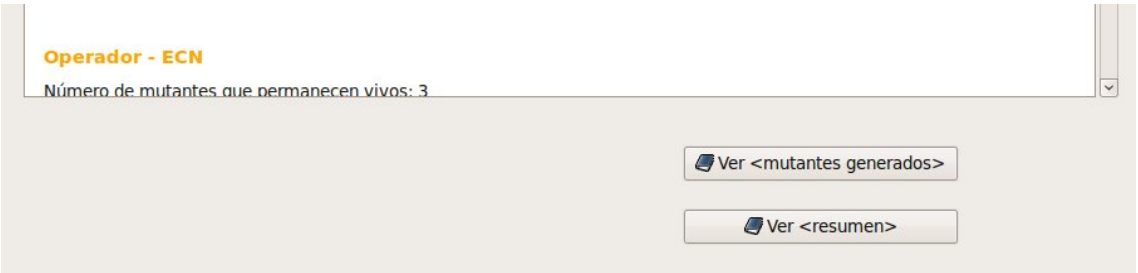

Figura 4.22: Mejora VII

- Para que le resultase más cómo al usuario y al mismo tiempo le sirve como modo de información de los resultados, se dispuso la forma de escoger el fichero mutante a comparar mediante desplegables, en el que primero se indicaba el estado, luego el operador que esté en ese estado y finalmente la instrucción y atributo de dicho

operador en el estado seleccionado.

- Conseguimos limpieza en la comparación de código, si a la hora de seleccionar un estado escogemos "Estado", de este modo se limpiará la pantalla.
- Mostrando las diferencias entre los ficheros mutantes y WS-BPEL, coloreamos las líneas dispares en rojo en el código mutante y en verde en el código original. De este modo indicamos con el rojo dónde se ha equivocado el usuario y en verde cómo debería de escribirse correctamente.

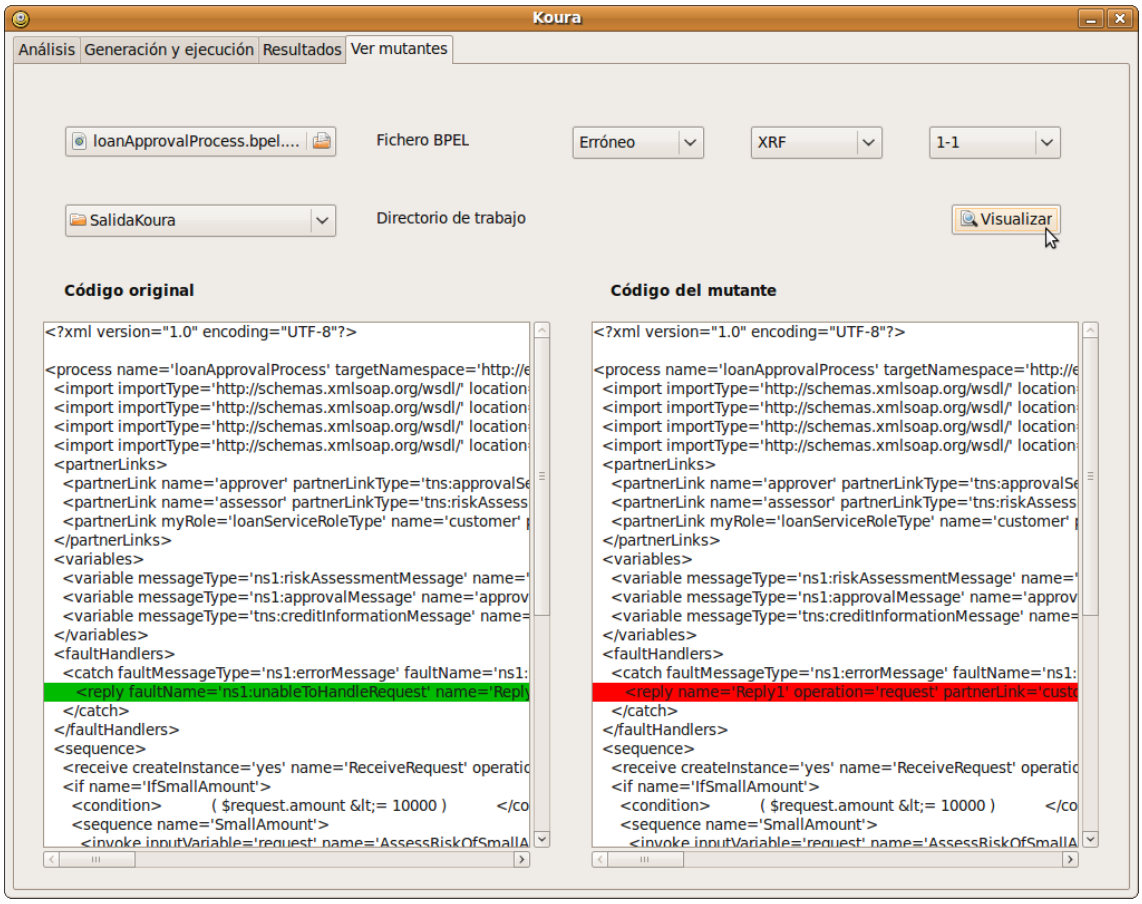

Figura 4.23: Mejora VIII, IX y X

4.3. Diseño del sistema

Durante el proceso de análisis del sistema hemos obtenido la interacción y funciones del sistema, el modelo conceptual, las respuestas del sistema y operaciones del mismo. Durante el diseño obtendremos la arquitectura, clases softwares y operaciones.

El diseño del software forma parte del núcleo técnico del proceso de ingeniería del software y se aplica independientemente del paradigma de desarrollo utilizado. A partir del análisis y especificación de los requisitos del sofware, el diseño es la primera de las tres actividades técnicas (diseño, codificación y prueba) que hay que realizar para construir y verificar el software. Es un proceso iterativo a través del cual se traducen los requisitos en un modelo o representación del sistema que se va a construir. Inicialmente, el diseño se representa a un alto nivel de abstracción y a medida que se realizan iteraciones, el refinamiento siguiente lleva a representaciones del diseño de mucho menor nivel de abstracción.

4.3.1. Arquitectura del sistema software

En el apartado presente daremos una descripción de los subsistemas existentes y de los componentes con sus relaciones del sistema software. Dentro de la determinación de la arquitectura software necesitaremos establecer qué propiedades ha de cumplir el sistema y de qué recursos disponemos.

La arquitectura del software es la estructura del sistema, componentes, propiedades y relaciones.

En nuestra aplicación podemos encontrar una división en etapas de la arquitectura.

1. Capa de interfaz de usuario: Incluye todos los componentes utilizados para la representación de la interfaz de usuario de la aplicación.

2. Capa de control de la aplicación: Incluye componentes y procedimientos que permiten el control de eventos y mandar mensajes a las capas necesarias.

3. Capa de dominio: Incluye todas las clases con las que trabaja el sistema.

4.3.2. Diagramas de secuencia

Mediante los diagramas de secuencia definiremos la interacción entre las clases de objetos en respuesta a los eventos producidos en el sistema. Utilizaremos los diagramas de secuencia ya que éstos dan a conocer de una forma efectiva el orden de los mensajes entre objetos y eventos. Utilizaremos cajas para representar objetos, clases y multiobjetos.

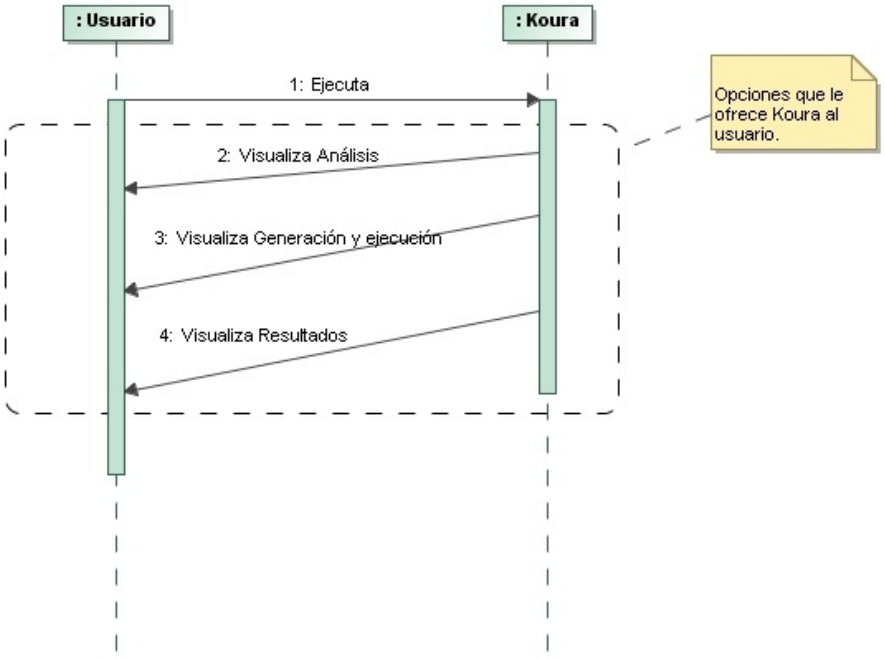

Figura 4.24: Diagrama de comportamiento - Inicio

Al ejecutar Koura, el usuario tiene la opción de visualizar varias pantallas, donde puede entrar de manera independiente, sin tener que realizar el recorrido desde el principio. Obviamente esto es posible si anteriormente se ha realizado el paso previo de la pestaña a la que se quiera acceder.

Tras entrar en Koura, el usuario dispone de las siguientes opciones:

1. Visualizar la pantalla de “Análisis”.
2. Visualizar la pantalla de “Generación y ejecución”.
3. Visualizar una de las pantallas de “Resultados”.

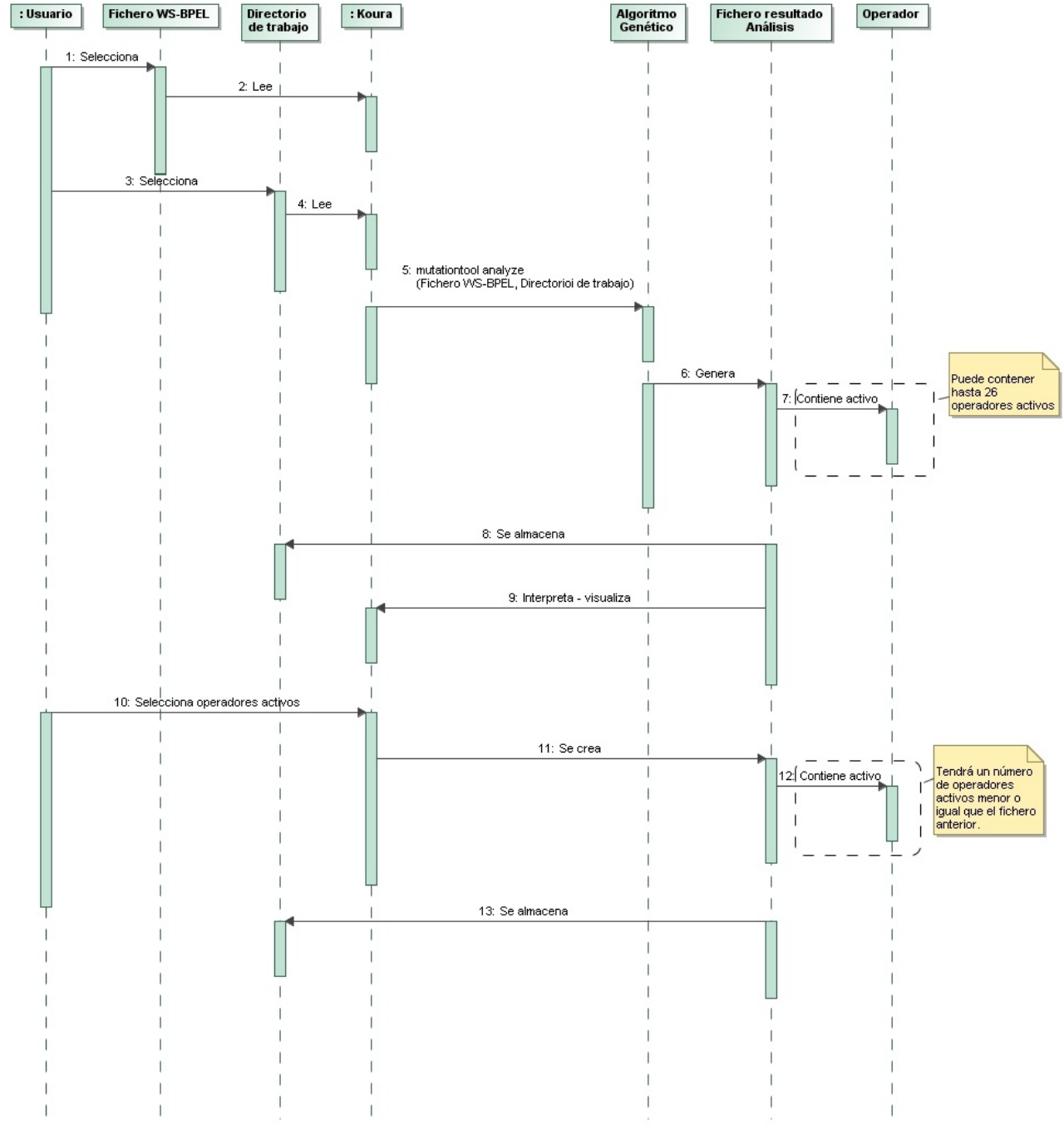

Figura 4.25: Diagrama de comportamiento - Análisis

El usuario tras seleccionar el fichero y el directorio con el que quiere realizar el estudio, le indica a Koura que realice el análisis. Koura llama al algoritmo genético y le devuelve los resultados del análisis que koura posteriormente interpreta.

El usuario tiene las opciones:

1. Trabajar en los siguientes pasos con todos los operadores que Koura le ha presentado como disponibles tras realizar el análisis.
2. Seleccionar un conjunto de los operadores que Koura le ha presentado como disponibles tras realizar el análisis.

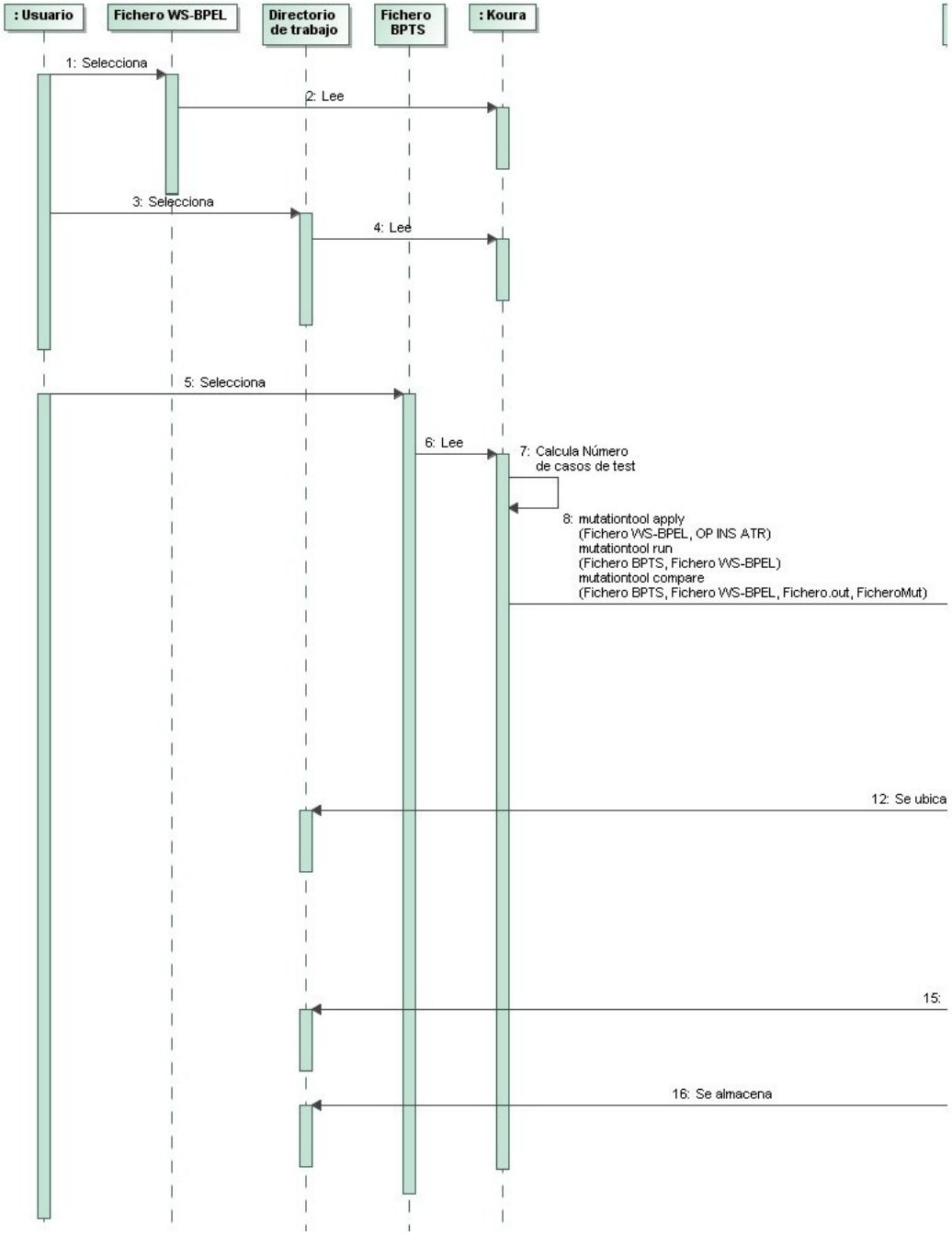

Figura 4.26: Diagrama de comportamiento - Generación y ejecución: Todos (parte 1)

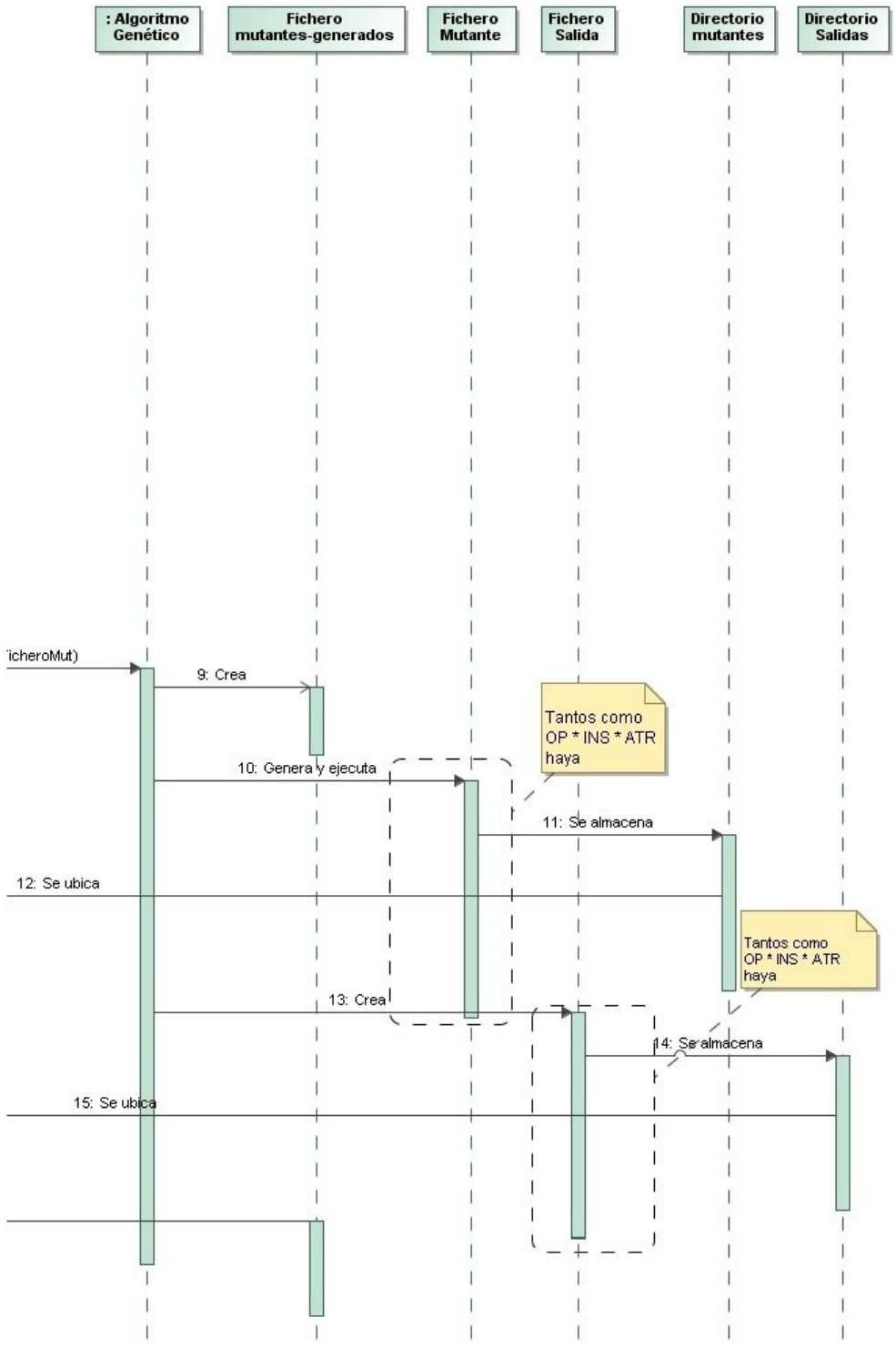

Figura 4.27: Diagrama de comportamiento - Generación y ejecución: Todos (parte 2)

El usuario tras haber solicitado realizar el análisis del fichero WS-BPEL seleccionado, quiere realizar la generación y ejecución de todos los mutantes disponibles con dicho estudio. Éste se lo hace saber a Koura introduciendo los parámetros necesarios y Koura realiza la llamada al algoritmo genético y éste le devuelve todos los ficheros resultantes de las operaciones realizadas.

El usuario cuando se encuentra en la pantalla de "Generación y ejecución", tiene las siguientes opciones:

1. Realizar la generación y ejecución con todos los operadores.
2. Realizar la generación y ejecución con una selección de los operadores.

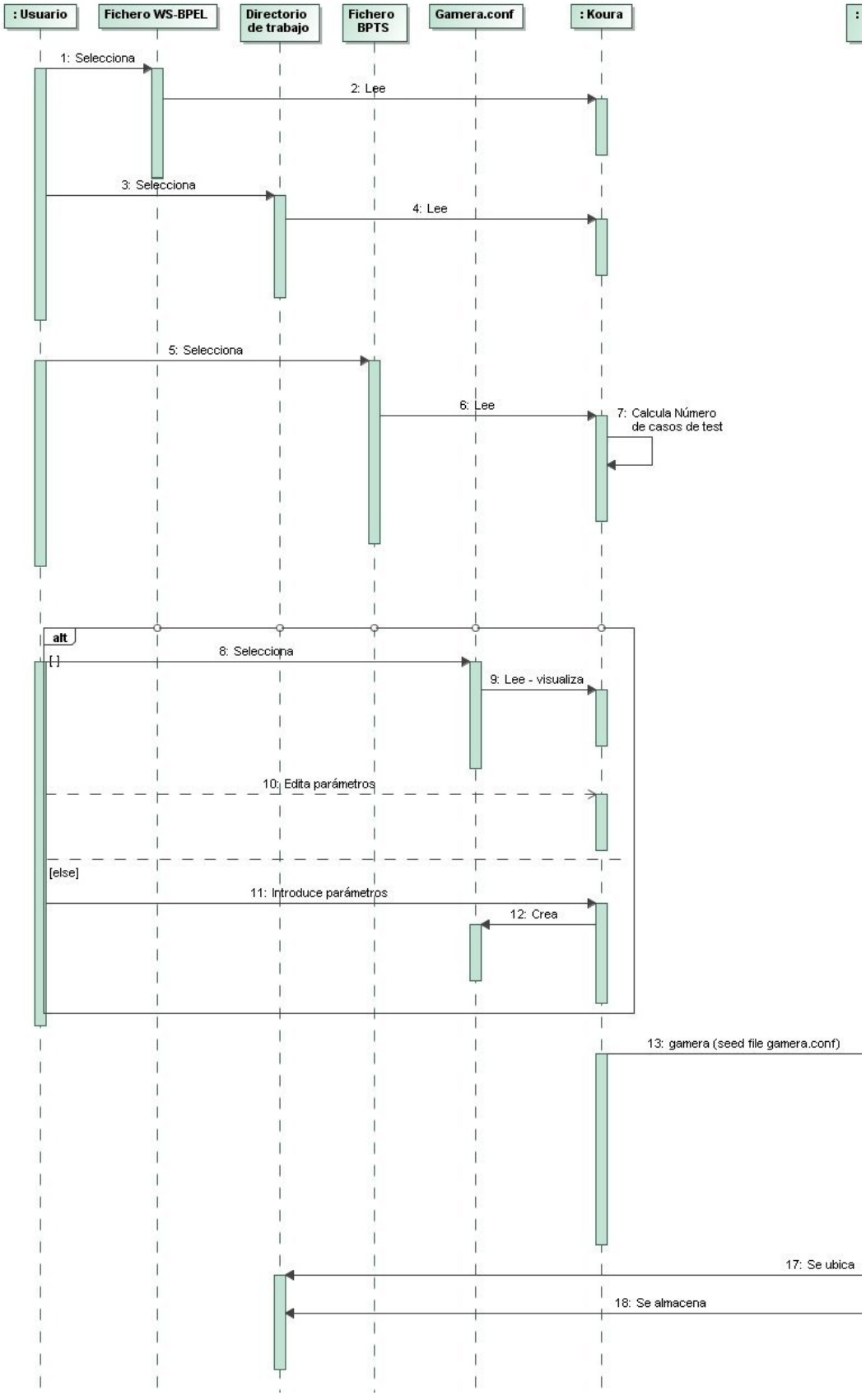

Figura 4.28: Diagrama de comportamiento - Generación y ejecución: Selección (parte 1)

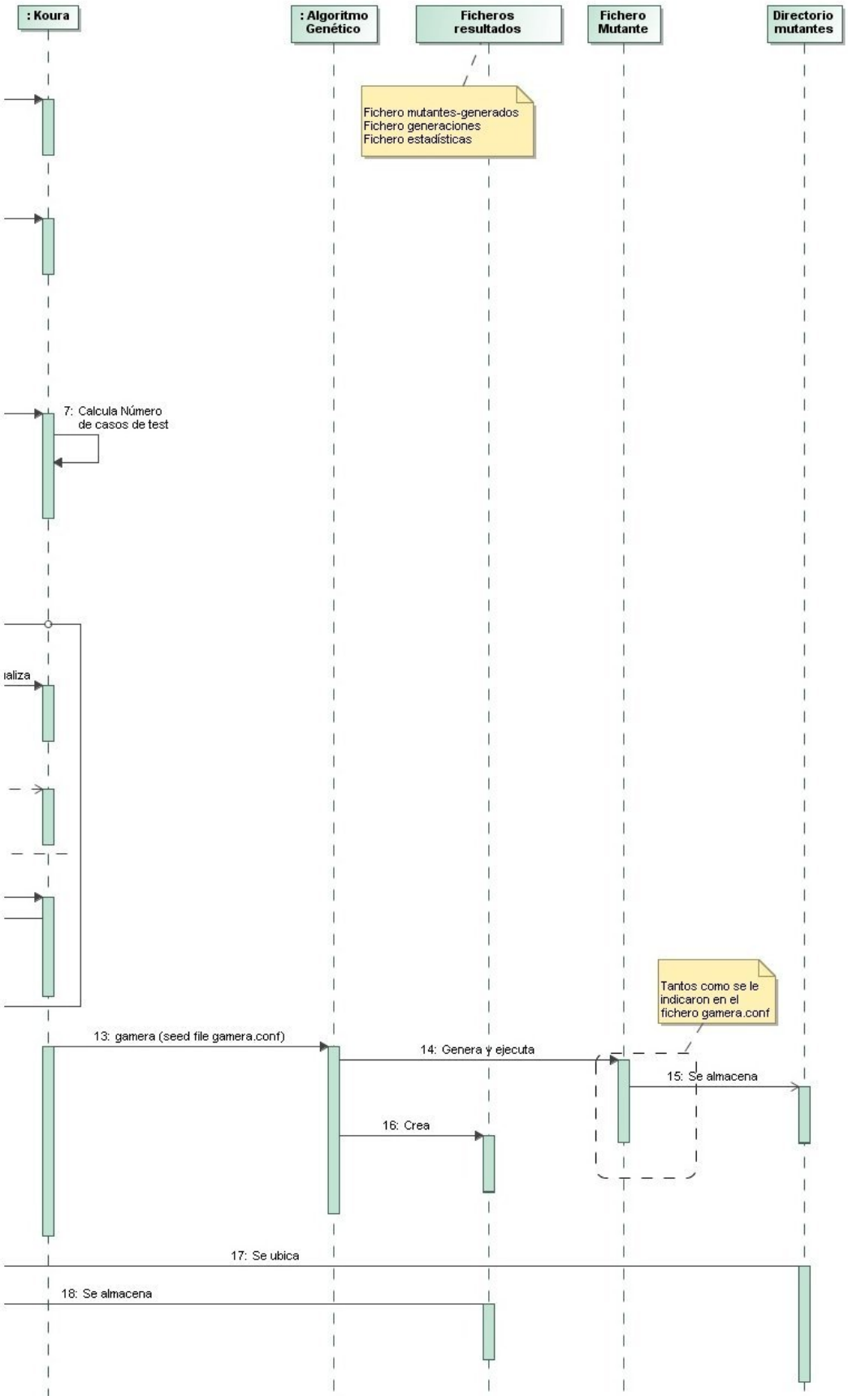

Figura 4.29: Diagrama de comportamiento - Generación y ejecución: Selección (parte 2)

El usuario tras haber solicitado realizar el análisis del fichero WS-BPEL seleccionado, quiere realizar la generación y ejecución de una sección de los mutantes disponibles con dicho estudio. Éste se lo hace saber a Koura introduciendo los parámetros necesarios y Koura realiza la llamada al algoritmo genético y éste le devuelve todos los ficheros resultantes de las operaciones realizadas.

El usuario puede:

1. Seleccionar los parámetros necesarios para realizar la generación y ejecución mediante un fichero gamera.conf, que cargará la configuración en pantalla.
   a) Puede dejar la configuración tal y como se le presenta.
   b) Puede modificar algunos parámetros de la configuración mostrada.
2. Introducir cada uno de los parámetros a mano según sus necesidades en el estudio.

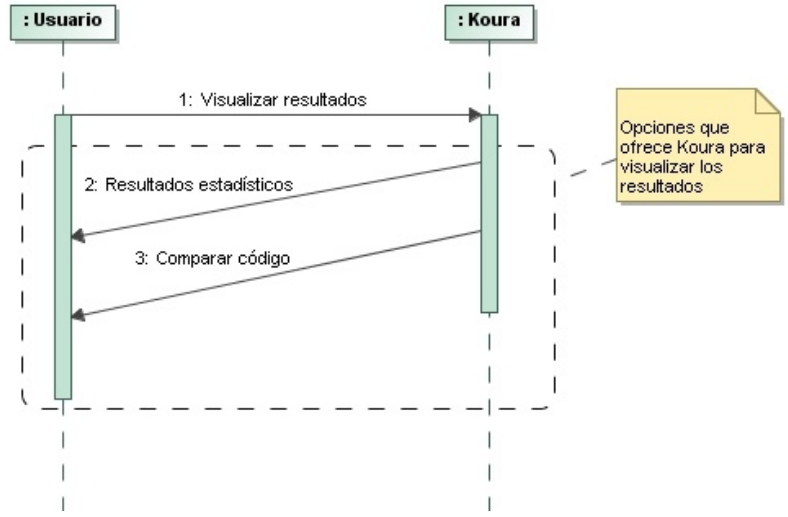

Figura 4.30: Diagrama de comportamiento - Resultados

El usuario después de realizar el estudio de Análisis + Generación + Ejecución quiere visualizar los resultados obtenidos.

Koura le ofrece al usuario dos opciones:

1. Visualizar los resultados mediante unos valores “estadísticos”.
2. Visualizar mediante la comparativa de código entre el fichero original WS-BPEL y un fichero mutante.

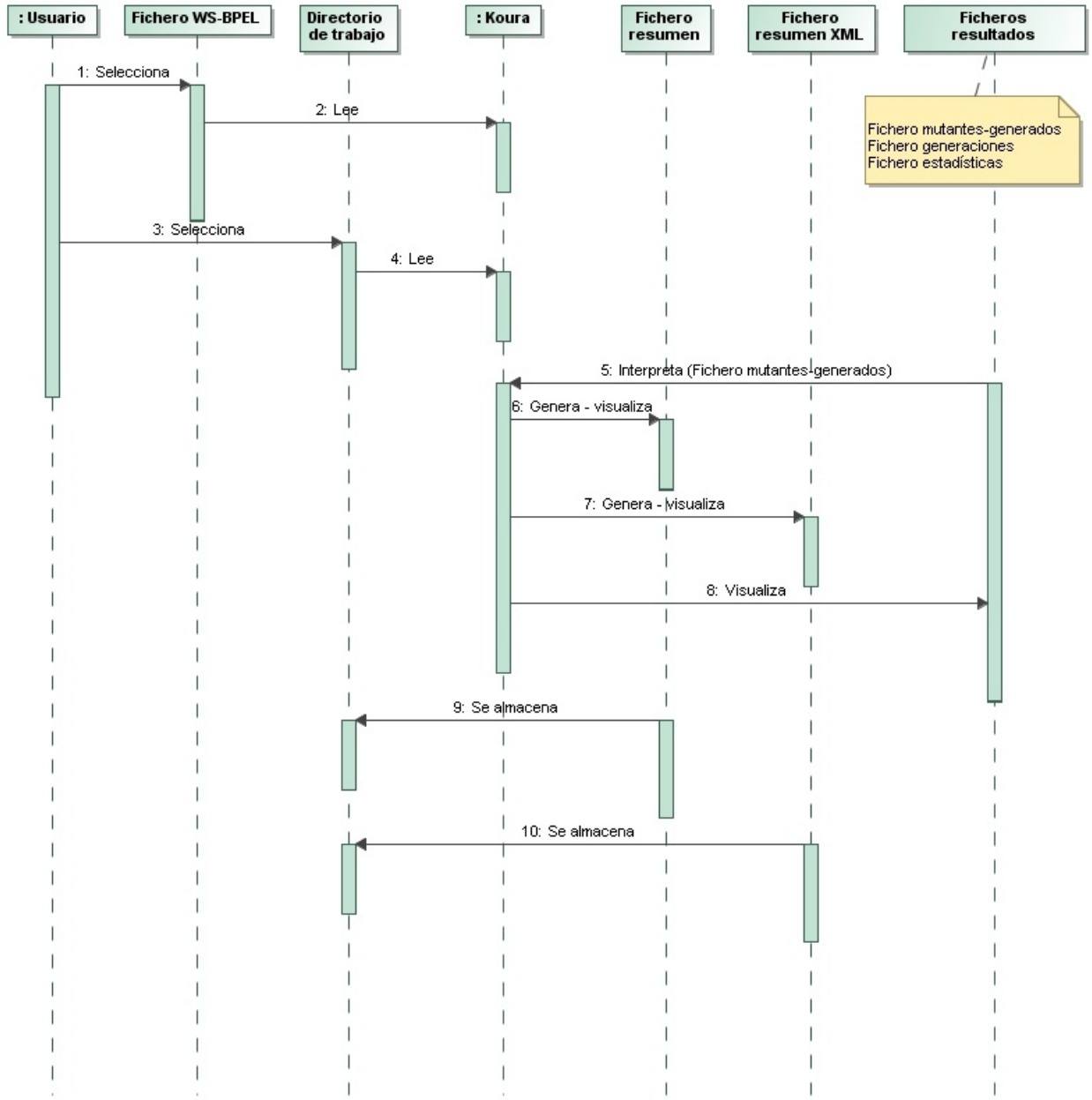

Figura 4.31: Diagrama de comportamiento - Resultados: Resultados estadísticos

El usuario quiere visualizar los resultados mediante los datos “estadísticos”, como viene de realizar el análisis + generación + ejecución, el fichero y el directorio de trabajo ya están seleccionados, en el caso de que desee cambiar, podrá realizarlo indicándoselo a Koura. Se interpretan los resultados obtenidos y se muestran por pantalla, creando de forma adicional un fichero resumen que engloba los valores importantes del estudio.

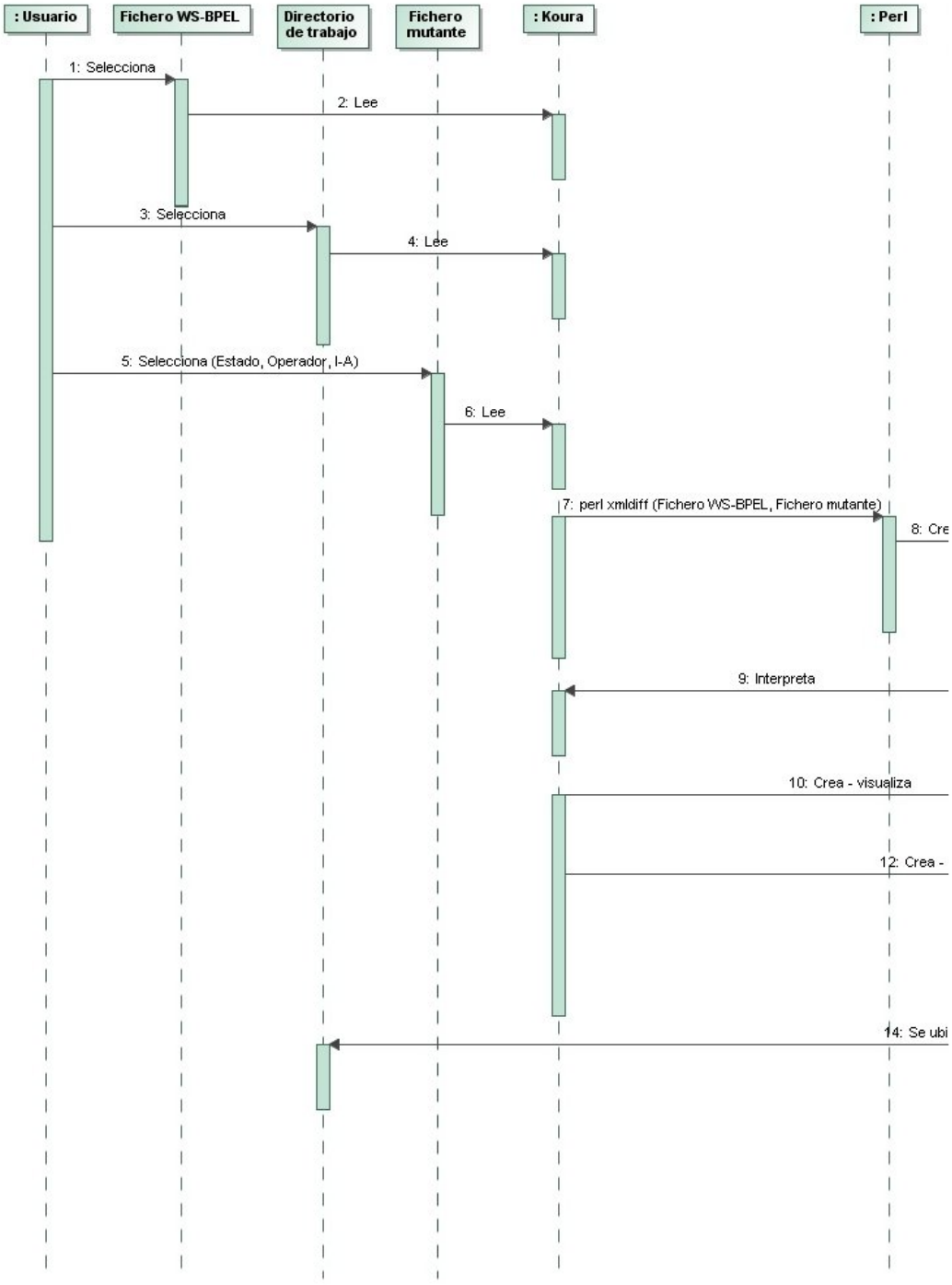

Figura 4.32: Diagrama de comportamiento - Resultados: Ver Mutantes (parte 1)

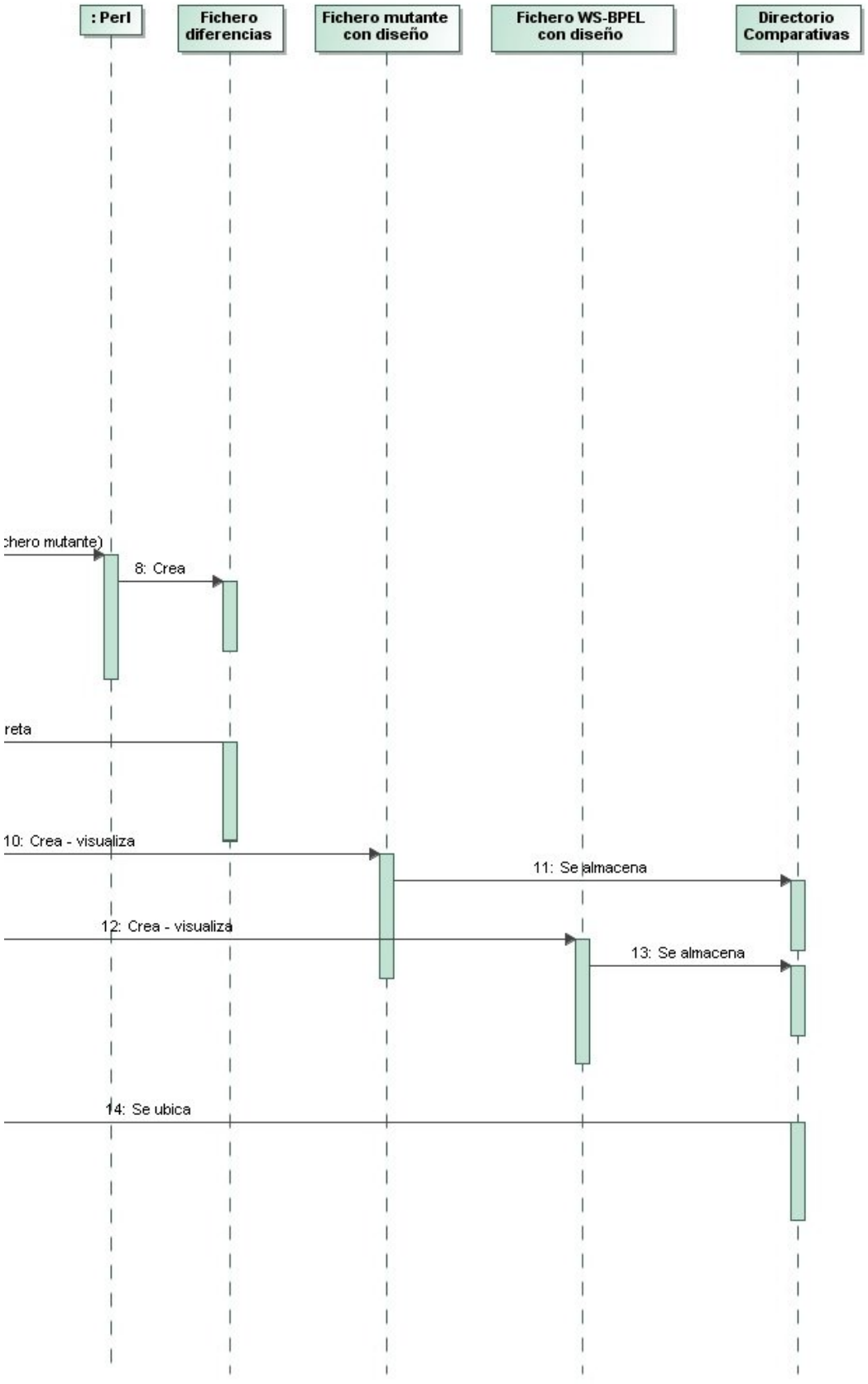

Figura 4.33: Diagrama de comportamiento - Resultados: Ver Mutantes (parte 2)

El usuario desea comparar los códigos del fichero WS-BPEL y el fichero mutante. Seleccionará en Koura el fichero mutante según el estado (vivo, muerto o error), el operador (se visualizarán en Koura los operadores disponibles según el estado seleccionado por el usuario) y el detalle de los valores de la instrucción y atributo que tenga dicho operador. Tras esto Koura lanzará un script de Perl que nos dará las diferencias entre los mutantes, este fichero de diferencias lo interpretará Koura y coloreará en pantalla las líneas dispares entre los ficheros.

El usuario tiene las opciones:

1. Seleccionar un nuevo estado.

2. Seleccionar un nuevo operador, si existe, que esté en el estado ya seleccionado.

3. Seleccionar un nuevo detalle de valores, si existen, de instrucción y atributo (I-A) del estado y operador seleccionados con anterioridad.

4.4. Pruebas y validación

Son múltiples los resultados que nos ofrece Koura según las opciones que seleccionemos a lo largo del estudio de mutación que realicemos. Nos vamos a centrar en dos de ellos: El tiempo obtenido tras la generación y ejecución de mutantes tanto si escogemos la opción de generar y ejecutar todos o una selección de los mismos (siempre pondríamos un mismo porcentaje de selección) y la comparativa de código, comprobaremos que efectivamente obtenemos los resultados esperados según el operador mutante escogido.

4.4.1. El tiempo

A la hora de las pruebas del tiempo, tenemos que tener en cuenta que si el ActiveBPEL no está activado dicha prueba va a tardar más que si ya contamos con que éste ya está activado, el tiempo que se tarda en que se active también lo tenemos en consideración a la hora de medir el tiempo.

Primera prueba

Vamos a generar todos los mutantes con el caso de prueba LoanApproval y el ActiveBPEL no está activado, veamos los resultados:

Tiempo = 419.14 seg.

Segunda prueba

Volvemos a generar todos los mutantes con el caso de prueba LoanApproval y el ActiveBPEL está activado (debido a la prueba anterior), veamos los resultados:

Tiempo = 363.51 seg.

Tercera prueba

Ahora generaremos un 20 % de los mutantes con el caso de prueba LoanApproval y el ActiveBPEL no está activado, veamos los resultados:

Tiempo = 64.258 seg.

Cuarta prueba

Generamos de nuevo un 20 % de los mutantes con el caso de prueba LoanApproval y el ActiveBPEL está activado (de la prueba anterior), veamos los resultados:

Tiempo = 30.373 seg.

Quinta prueba

Generamos todos los mutantes con el caso de prueba SquaresSum y el ActiveBPEL no está activado, veamos los resultados:

Tiempo = 281.56 seg.

Sexta prueba

Generamos todos los mutantes con el caso de prueba SquaresSum y el ActiveBPEL está activado (de la prueba anterior), veamos los resultados:

Tiempo = 236.81 seg.

Séptima prueba

Generamos un 20 % de los mutanes con el caso de prueba SquaresSum y el ActiveBPEL no está activado, veamos los resultados:

Tiempo = 78.434 seg.

Octava prueba

Generamos un 20 % de los mutantes con el caso de prueba SquaresSum y el ActiveBPEl está activado (de la prueba anterior), veamos los resultados:

Tiempo = 40.604 seg.

Novena prueba

Generamos todos los mutantes con el caso de prueba Marketplace y el ActiveBPEl no está activado, veamos los resultados:

Tiempo = 534.26 seg.

Décima prueba

Generamos todos los mutantes con el caso de prueba MarketPlace y el ActiveBPEL está activado (de la prueba anterior), veamos los resultados:

Tiempo = 494.96 seg.

Undécima prueba

Generamos un 20 % de los mutanes con el caso de prueba MarketPlace y el ActiveBPEL no está activado, veamos los resultados:

Tiempo = 192.92 seg.

Duodécima prueba

Generamos un 20 % de los mutantes con el caso de prueba MarketPlace y el ActiveBPEl está activado (de la prueba anterior), veamos los resultados:

Tiempo = 163.13 seg.

Conclusiones

Con solo estas pruebas, hemos podido comprobar que efectivamente sin estar activado el ActiveBPEL se tarda más en la generación y ejecución de mutantes.

En estas pruebas la mínima diferencia entre un caso de prueba con ActiveBPEL activado y no es de 33.885 seg. y la máxima 55.63 seg.

Otros datos a tener en cuenta; tras realizar la generación y ejecución de todos los casos de prueba (generando un 20 % de los mutantes), vemos que el caso de prueba que más mutantes genera es MetaSearch y el que menos SquaresSum.

4.4.2. Las comparativas

Ahora vamos a validar que según el operador seleccionado por el usuario, Koura nos va a devolver el resultado esperado. Comprobaremos los operadores mutantes y a través de Koura visualizando los resultados. Veamos la siguiente tabla:

Operador	Descripción	Visualización en Koura
	Mutación de Identificadores	
ISV	Sustituye el identificador de una variable por el de otra del mismo tipo	Tanto en el original como en el mutante aparecerá la misma línea coloreada
	Mutación de Expresiones	
EAA	Sustituye un operador aritmético por otro del mismo tipo	Tanto en el original como en el mutante aparecerá la misma línea coloreada
EEU	Elimina el operador - unario de cualquier expresión	Idem
ERR	Sustituye un operador relacional por otro del mismo tipo	Idem
ELL	Sustituye a un operador lógico por otro del mismo tipo	Idem
ECC	Sustituye un operador de camino por otro del mismo tipo	Idem
ECN	Modifica una constante numérica	Idem
EMD	Modifica una expresión de duración	Idem
EMF	Modifica una expresión de fecha límite	(*)

(*) De estos operadores todavía no disponemos de ejemplos para aplicarlos y ver las diferencias con el fichero original. Pero el comportamiento de éstos será semejante a los que tienen una descripción similar. Para aquellos que no comparten una descripción, habría que estudiar el ejemplo resultante. En Koura no hay problemas a la hora de realizar la comparativa, ya que está implementada de tal forma que irá comparando y coloreando todas aquellas diferencias sin importarle el orden en que se encuentren.

Operador	Descripción	Visualización en Koura
	Mutación de Actividades	
	Relacionados con la concurrencia	
ACI	Cambia el atributo *createInstance* de las actividades de recepción de mensajes a *no*	Tanto en el original como en el mutante aparecerá la misma línea coloreada
AFP	Cambia una actividad *forEach* secuencial a paralela	Tanto en el original como en el mutante aparecerán las dos mismas líneas coloreadas (apertura y cierre)
ASF	Cambia una actividad *secuence* por una actividad *flow*	Idem
AIS	Cambia el atributo *isolated* de un *scope* a *no*	Tanto en el original como en el mutante aparecerá la misma línea coloreada
	Mutación de Actividades	
	No concurrentes	
AIE	Elimina un elemento *elseif* o el elemento *else* de una actividad *if*	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
AWR	Cambia una actividad *while* por una *repeatUntil* y viceversa	Tanto en el original como en el mutante aparecerá la misma línea coloreada
AJC	Elimina el atributo *joinCondition* de cualquier actividad en la que aparezca	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
ASI	Intercambia el orden de dos actividades hijas de una actividad *sequence*	Aparecerá coloreada una línea, pero en el original aparecerá en una posición diferente a la del mutante
APM	Elimina un elemento *onMessage* de una actividad *pick*	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
APA	Elimina el elemento *onAlarm* de una actividad *pick* o de un manejador de eventos	Tanto en el original como en el mutante aparecerá la misma línea coloreada

Operador	Descripción	Visualización en Koura
	Mutación de Condiciones Excepcionales y Eventos	
XMF	Elimina un elemento *catch* o el elemento *catchAll* de un manejador de fallos	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
XRF	Elimina el atributo *faultName* de una actividad *reply*	Tanto en el original como en el mutante aparecerá la misma línea coloreada
XMC	Elimina la definición de un manejador de compensación	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
XMT	Elimina la definición de un manejador de terminación	(*)
XTF	Cambia el fallo lanzado por una actividad *throw*	(*)
XER	Elimina una actividad *rethrow*	(*)
XEE	Elimina un elemento *onEvent* de un manejador de eventos	(*)

Prueba ISV

El operador ISV, sustituye el identificador de una variable por el de otra del mismo tipo. Sólo se ve afectada una línea y como podemos comprobar esto es así. La diferencia está en que el nombre de la variable en el original es "counter" y en el mutante es "currentResult".

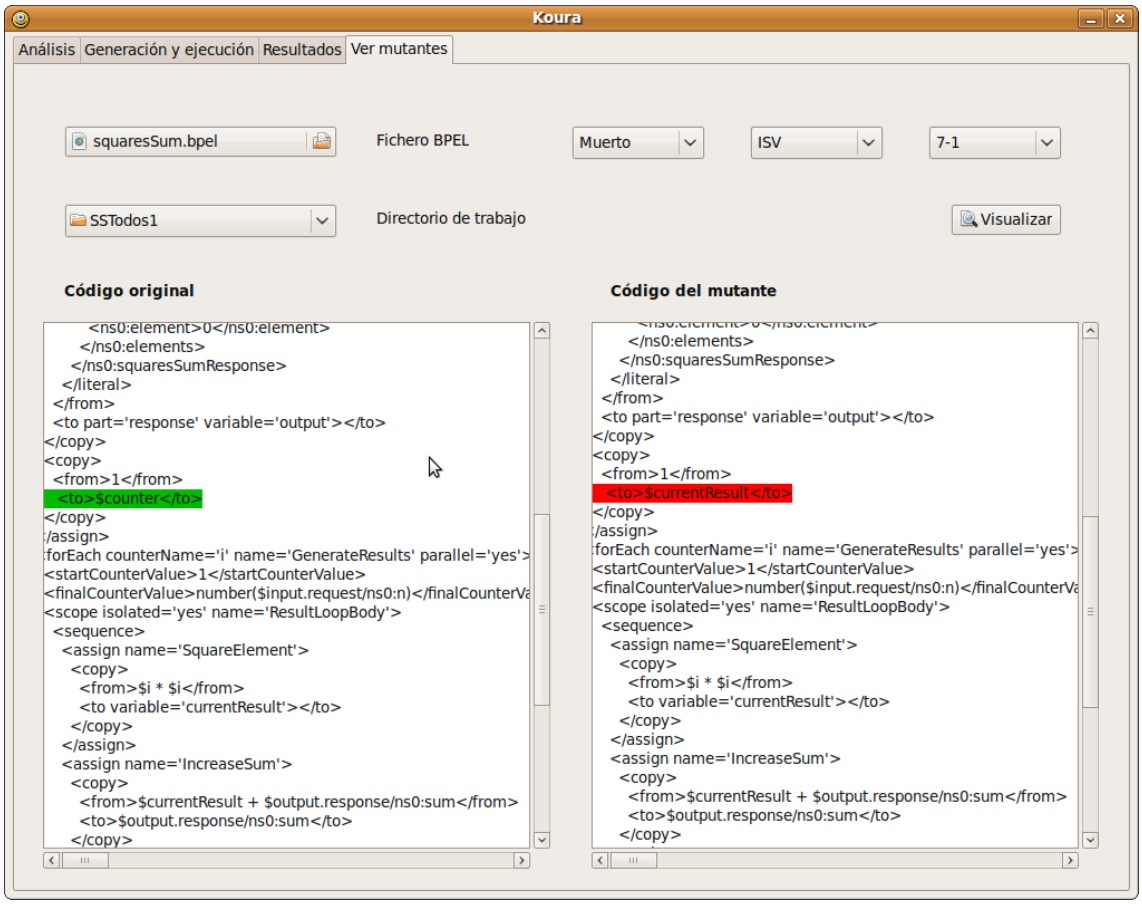

Figura 4.34: Operador ISV

Prueba EAA

El operador EAA, sustituye un operador aritmético por otro del mismo tipo. Se afecta una línea y esto se cumple. El operador que aparece en el fichero original es "*" y en el fichero mutante es "-".

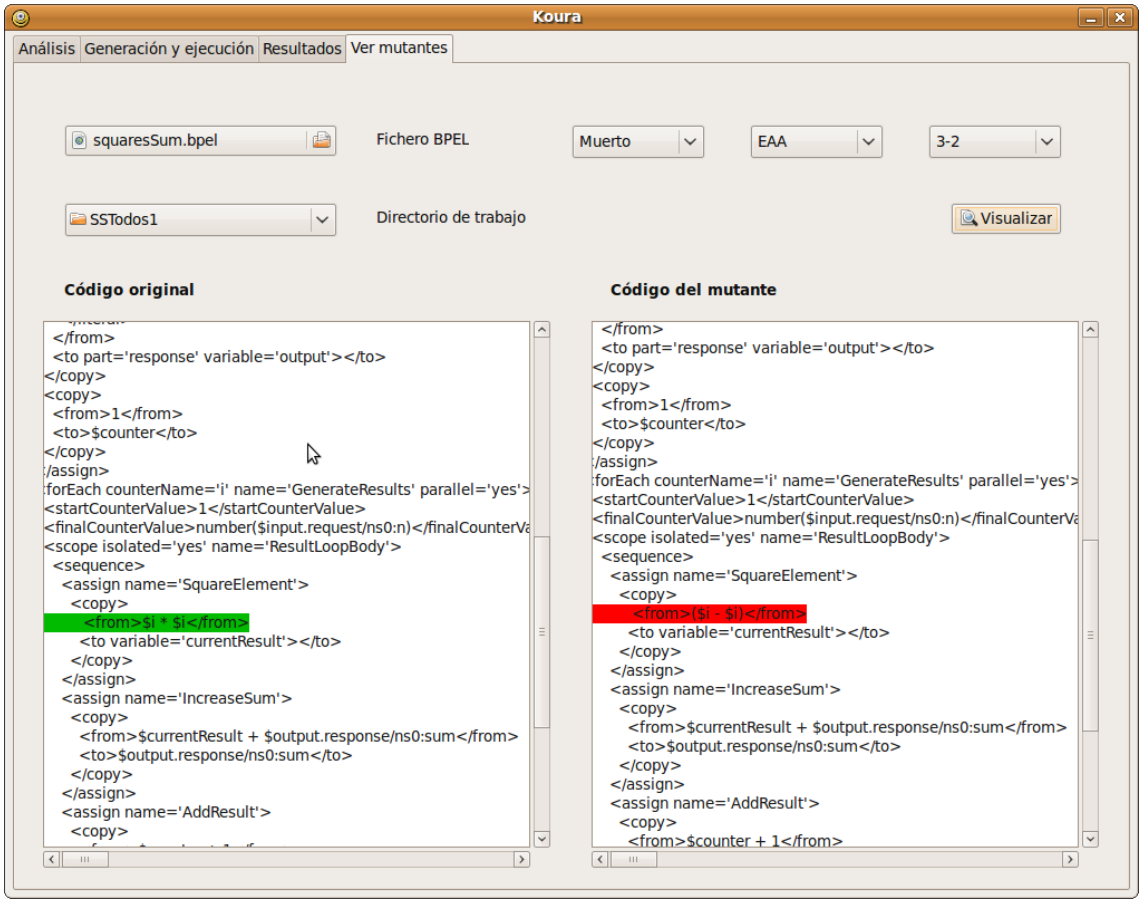

Figura 4.35: Operador EAA

Prueba ERR

El operador ERR, sustituye un operador relacional por otro del mismo tipo. Sólo tiene que señalarse una línea y como podemos comprobar esto se cumple. La diferencia está en que el valor en el original es 10000 y en el mutante es 10000.0.

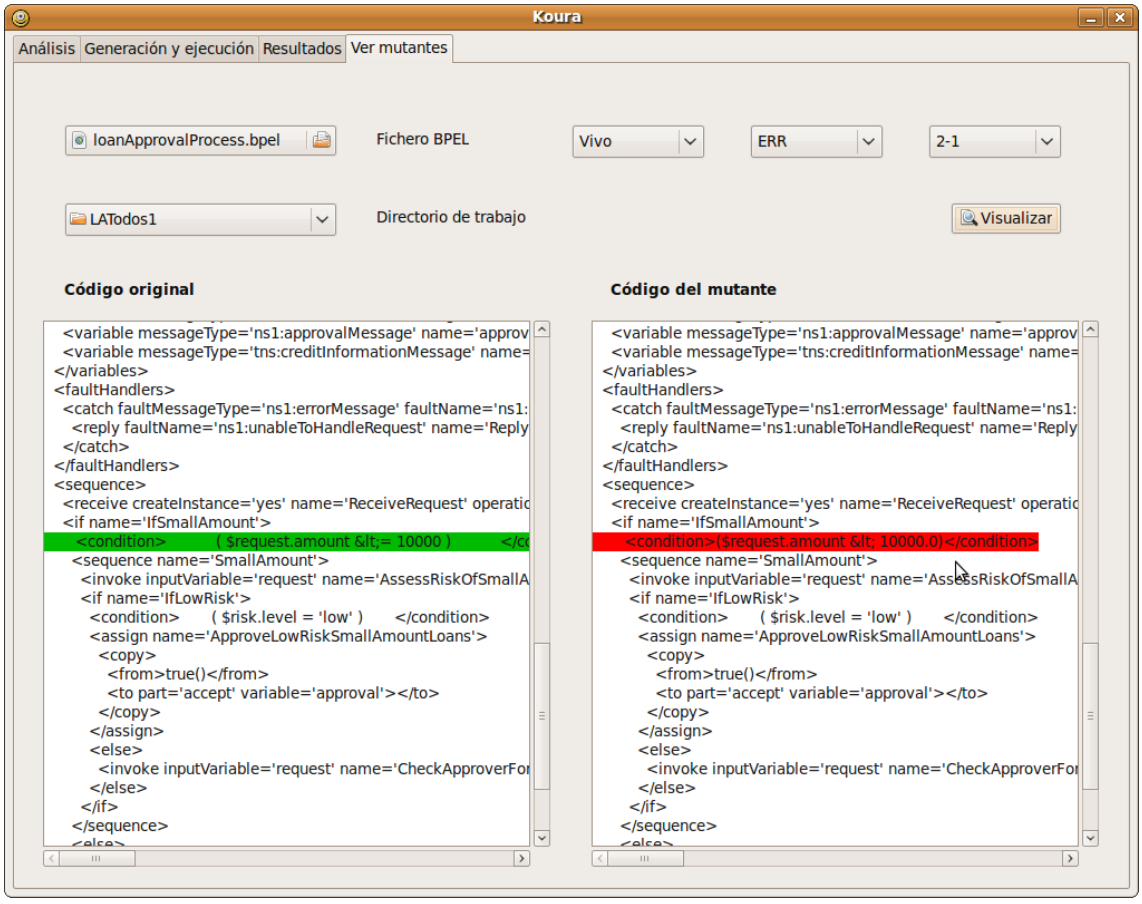

Figura 4.36: Operador ERR

Prueba ELL

El operador ELL, sustituye un operador lógico por otro del mismo tipo. Sólo se ve afectada una línea y podemos comprobar que esto es correcto.

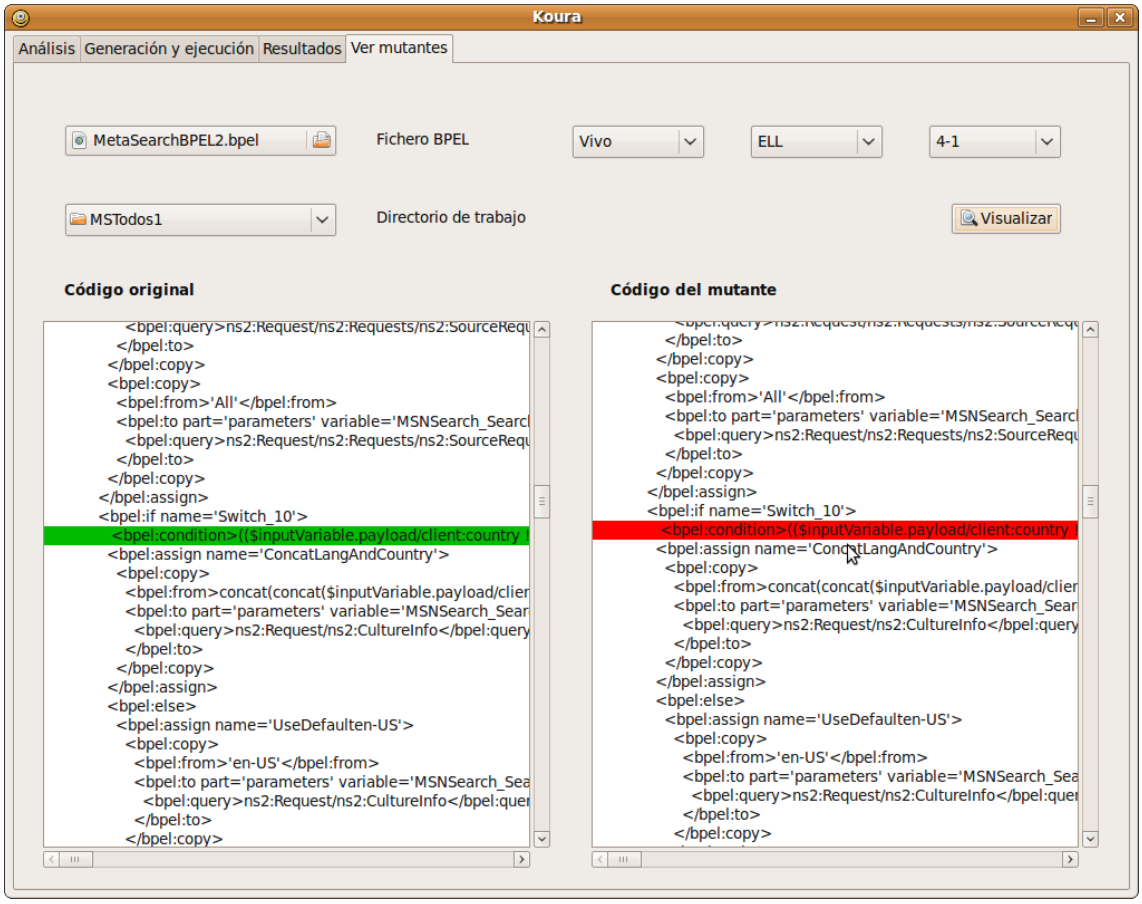

Figura 4.37: Operador ELL

Prueba ECC

El operador ECC, sustituye un operador de camino por otro del mismo tipo. Sólo afecta una línea y comprobamos que esto es así. En este caso en el fichero original, aparece en el operador de camino una sola "/" y en el mutante aparecen dos "//".

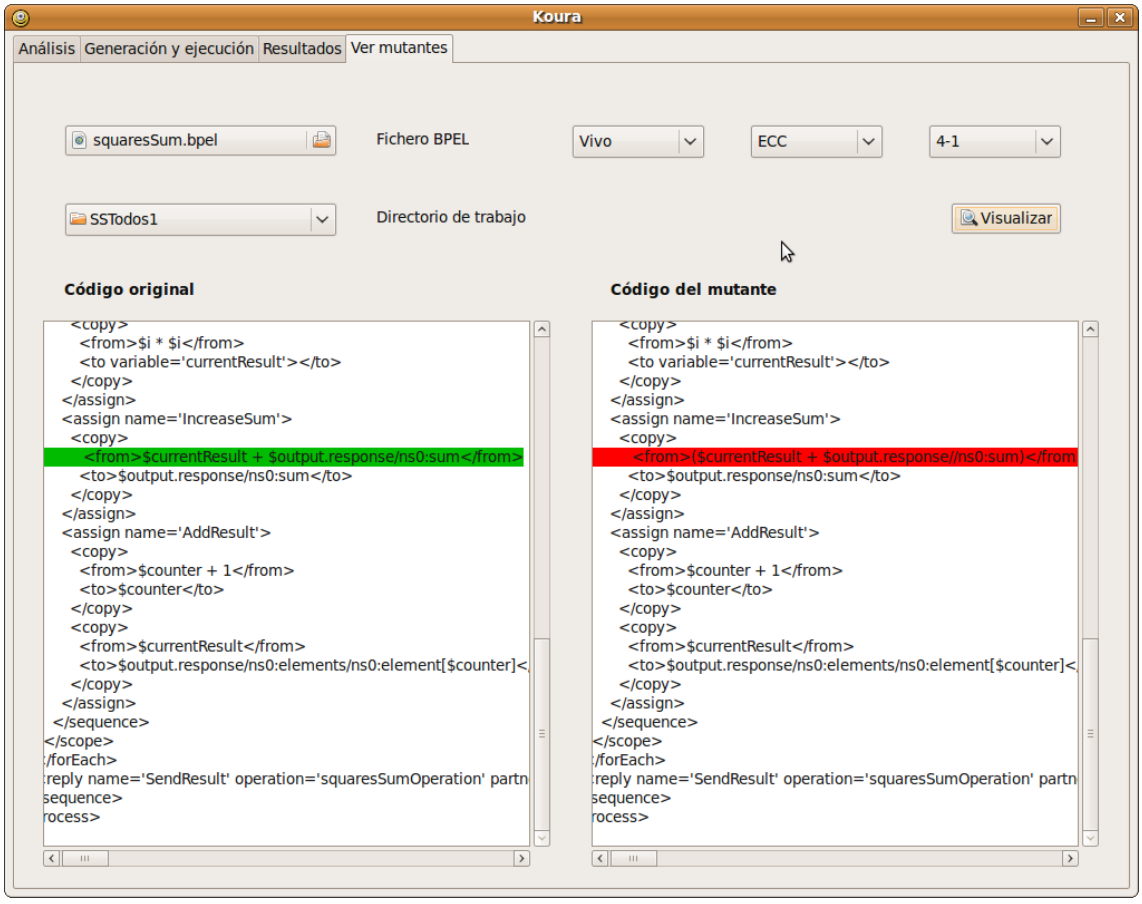

Figura 4.38: Operador ECC

Prueba ECN

El operador ECN, modifica una constante numérica. Se afecta una sola línea como podemos comprobar. En el fichero original el valor de la constante numérica es “10000” y en el mutante es “100000”.

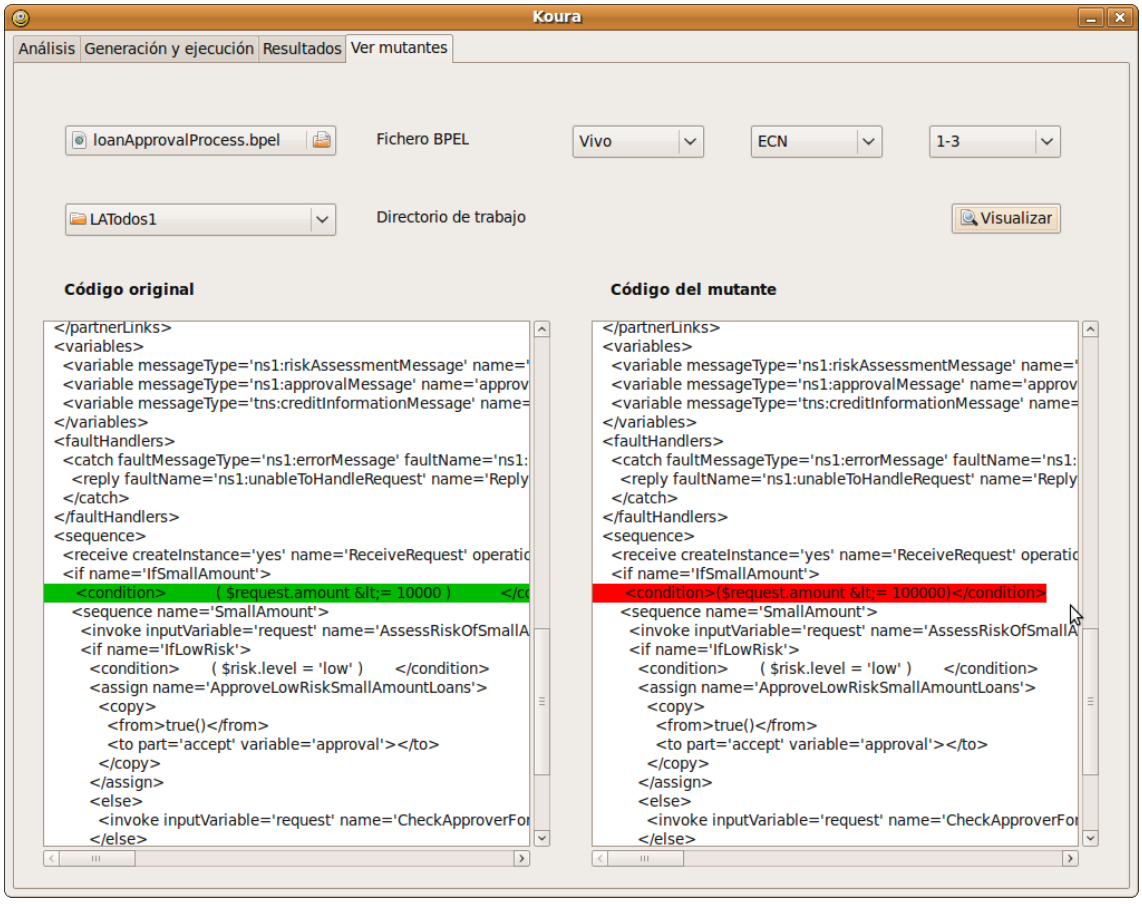

Figura 4.39: Operador ECN

Prueba EMD

El operador EMD, modifica una expresión de duración. Afecta a una línea como podemos comprobar. La expresión de duración en el fichero original es diferente a la del mutante.

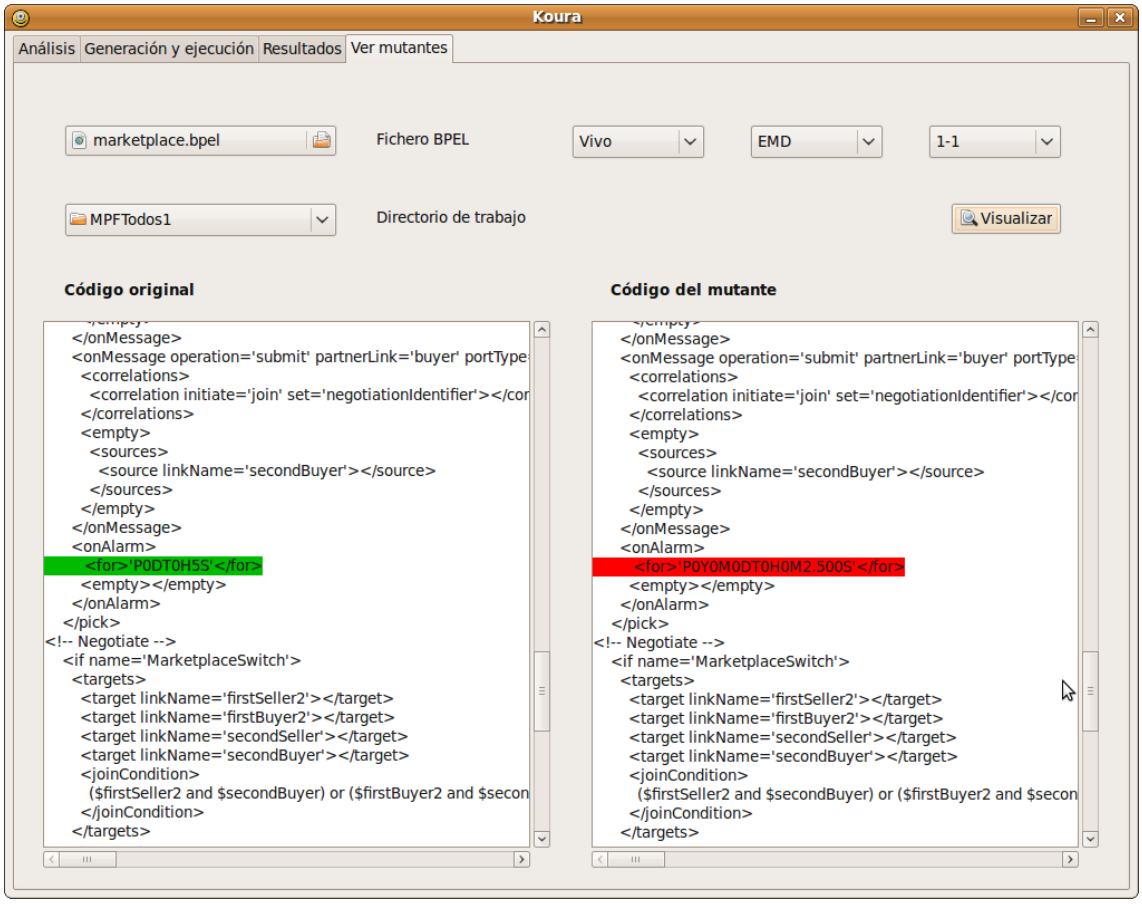

Figura 4.40: Operador EMD

Prueba ACI

El operador ACI, cambia el atributo *createInstance* de las actividades de recepción de mensajes a no. Hay una línea afectada como podemos comprobar. En el original aparece el atributo con valor “yes” y en el mutante con valor “no”.

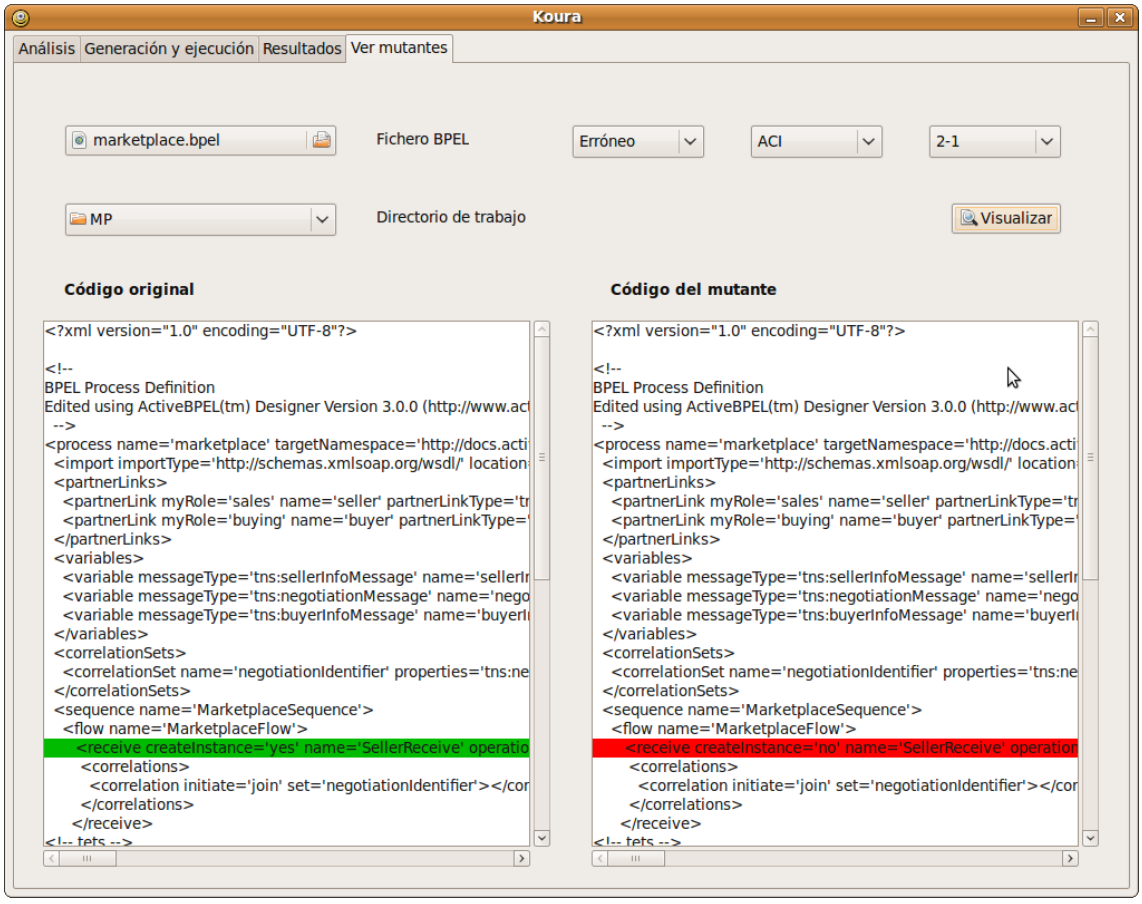

Figura 4.41: Operador ACI

Prueba AFP

El operador AFP, cambia una actividad *forEach* secuencial a paralela. Se ven afectadas dos líneas como podemos comprobar, la apertura y cierre de la actividad.

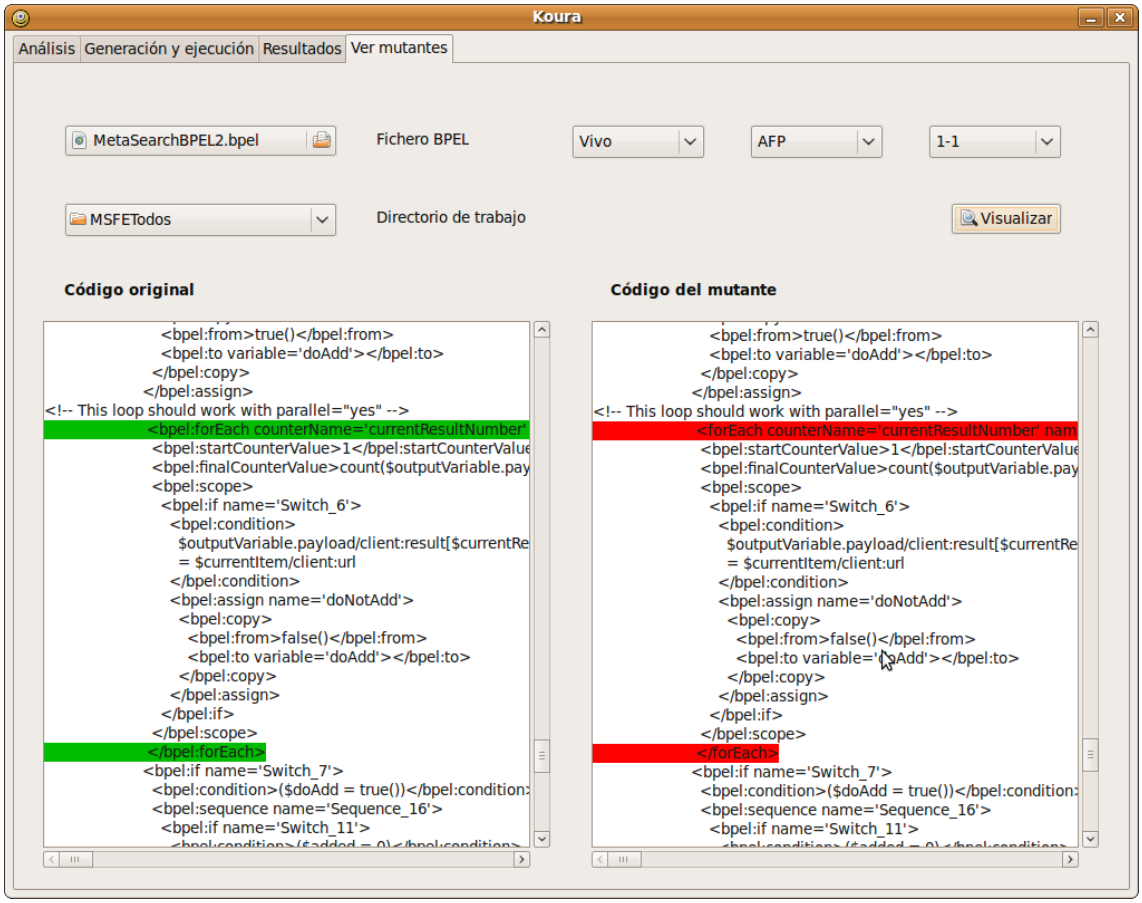

Figura 4.42: Operador AFP

Prueba ASF

El operador ASF, cambia una actividad *sequence* por una actividad *flow*. Se ven afectadas dos líneas tal y como vemos en el resultado. Son la apertura y cierre de la actividad, en el original está la actividad sequence y en el mutante la actividad flow con el resto de parámetros necesarios para la misma.

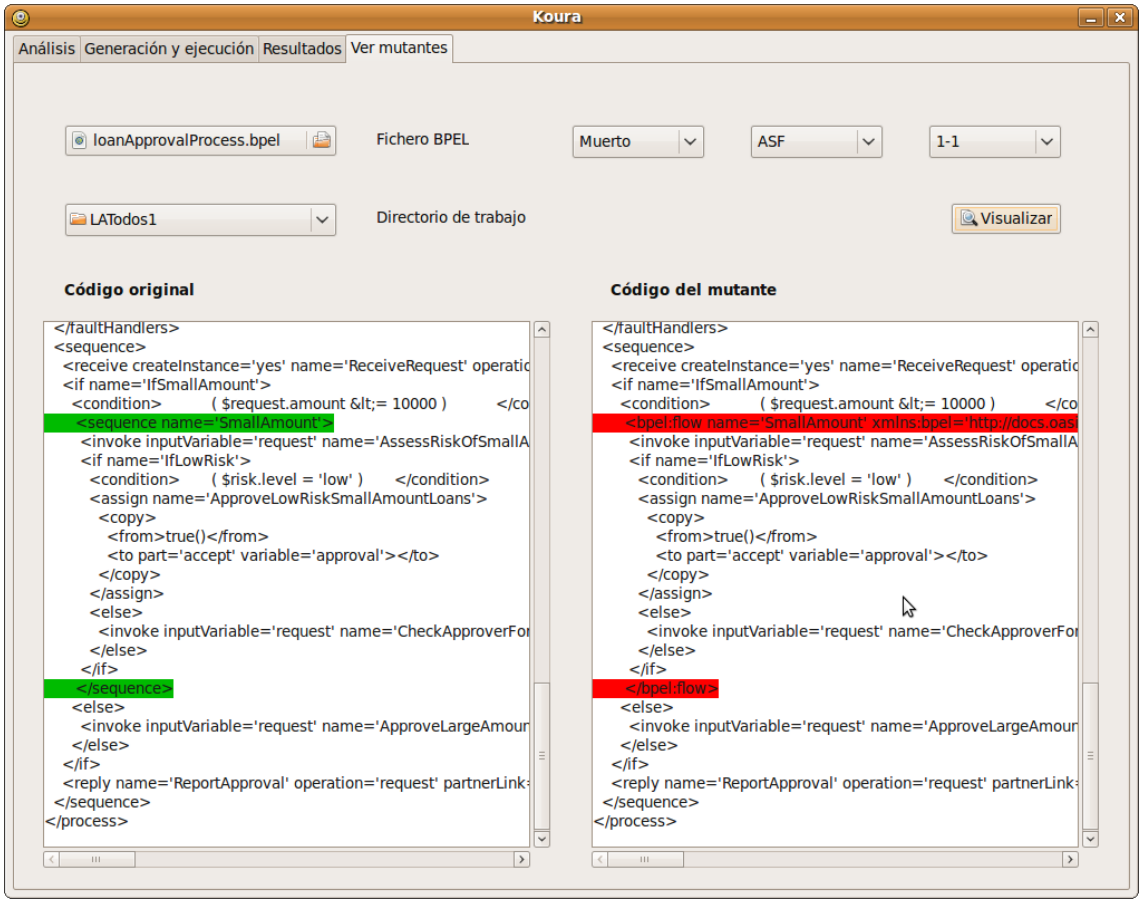

Figura 4.43: Operador ASF

Prueba AIS

El operador AIS, cambia el atributo *isolated* de un *scope* a no. Sólo la línea donde esté dicho atributo se ve afectada tal y comprobamos en el resultado. En el fichero original aparece con el valor en “yes” y en el mutante con el valor en “no”.

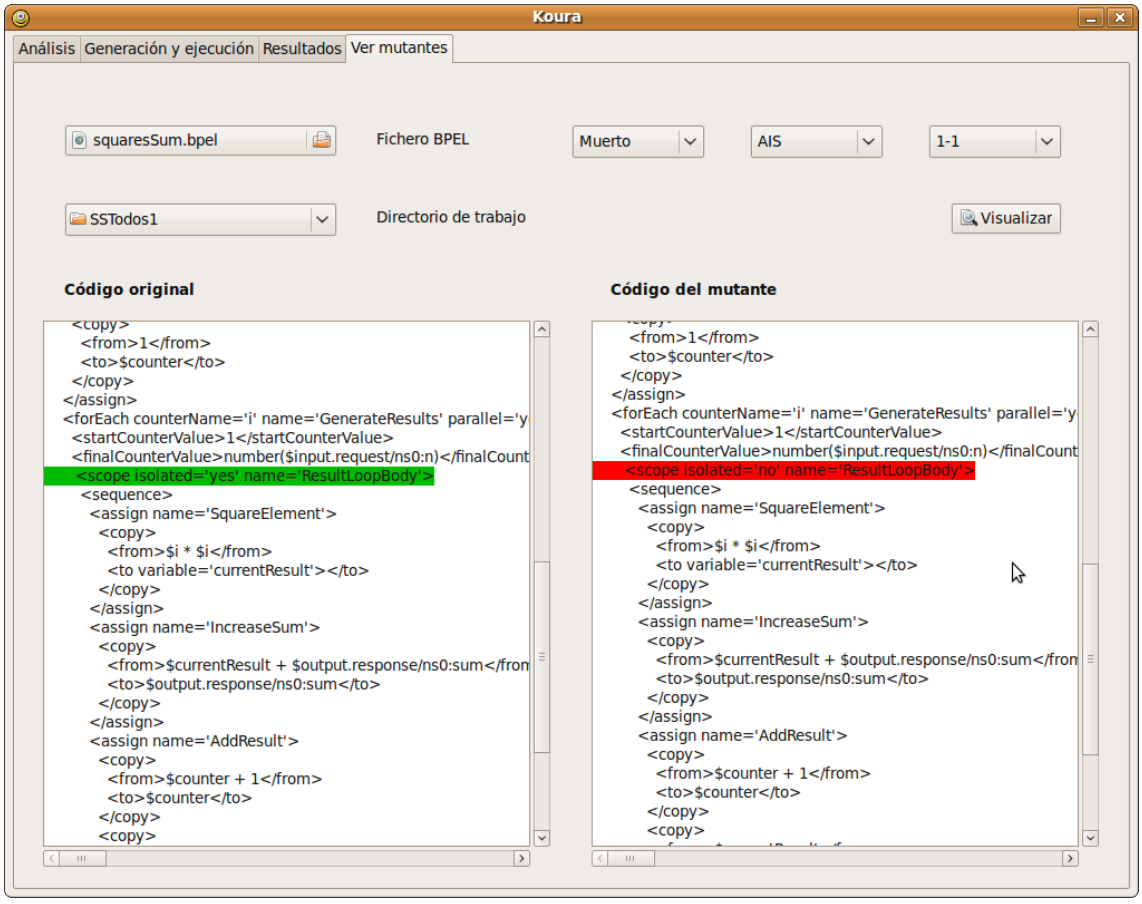

Figura 4.44: Operador AIS

Prueba AIE

El operador AIE, elimina un elemento *elseif* o el elemento *else* de una actividad *if*. Se ven afectadas todas las líneas de las que se componga dicho elemento, en este caso son 3, las cuales se pueden observar perfectamente en el fichero original y se ven en blanco en el fichero mutante.

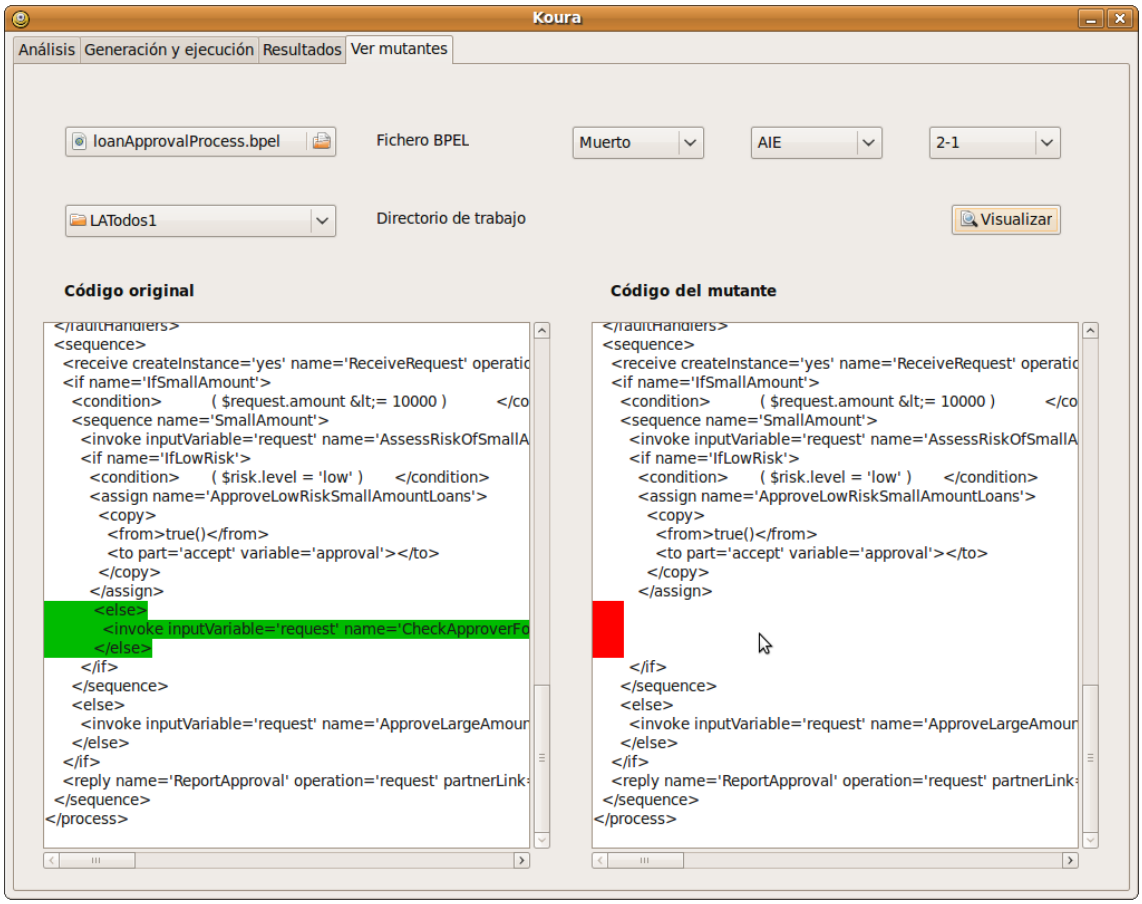

Figura 4.45: Operador AIE

Prueba AWR

El operador AWR, cambia una actividad *while* por una *repeatUntil* y viceversa. Se ven en este caso dos líneas afectadas, apertura y cierre de los bucles, en el original aparece el while y en el mutante repeatUntil.

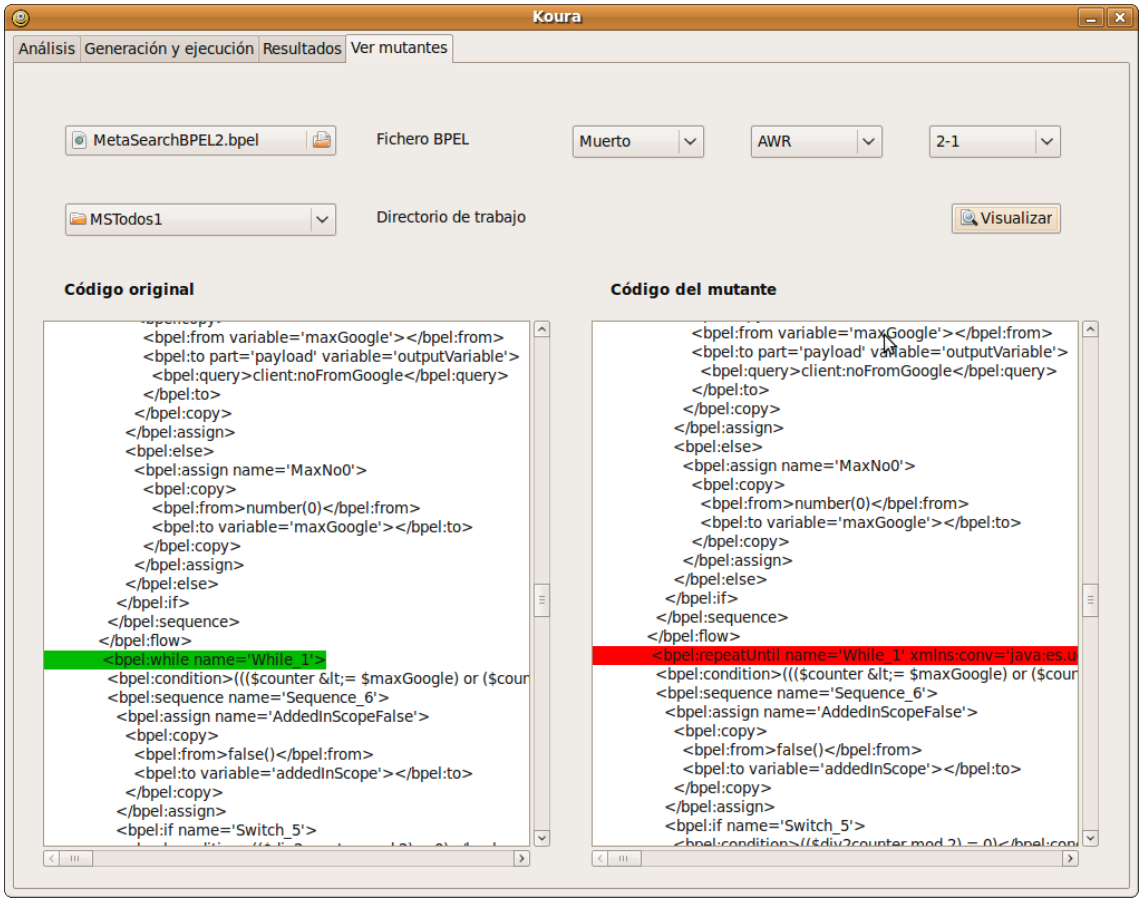

Figura 4.46: Operador AWR - apertura

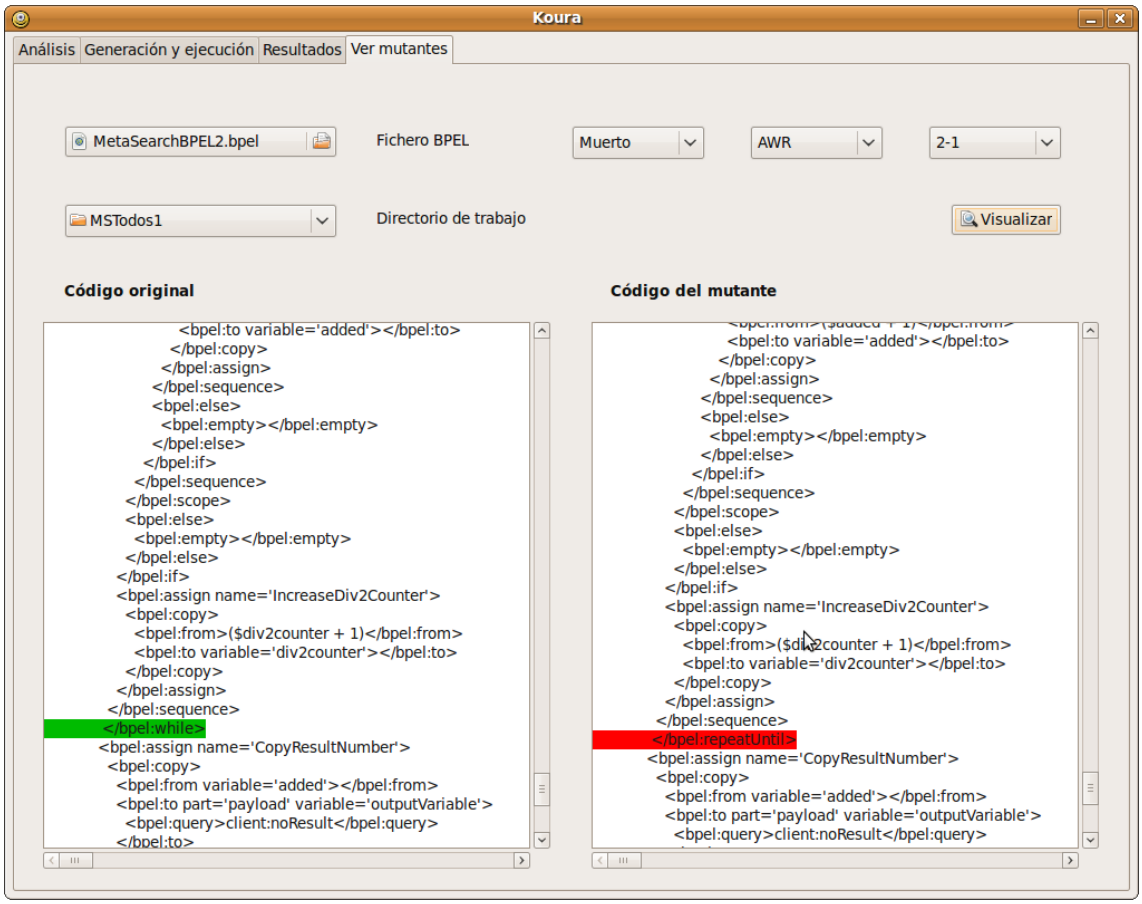

Figura 4.47: Operador AWR - cierre

Prueba ASI

El operador ASI, intercambia el orden de dos actividades hijas de una actividad *sequence*. Se ve afectada una línea como podemos comprobar, en el fichero original aparece cuando finaliza "if" y en el fichero mutante tiene lugar antes de que comience el "if".

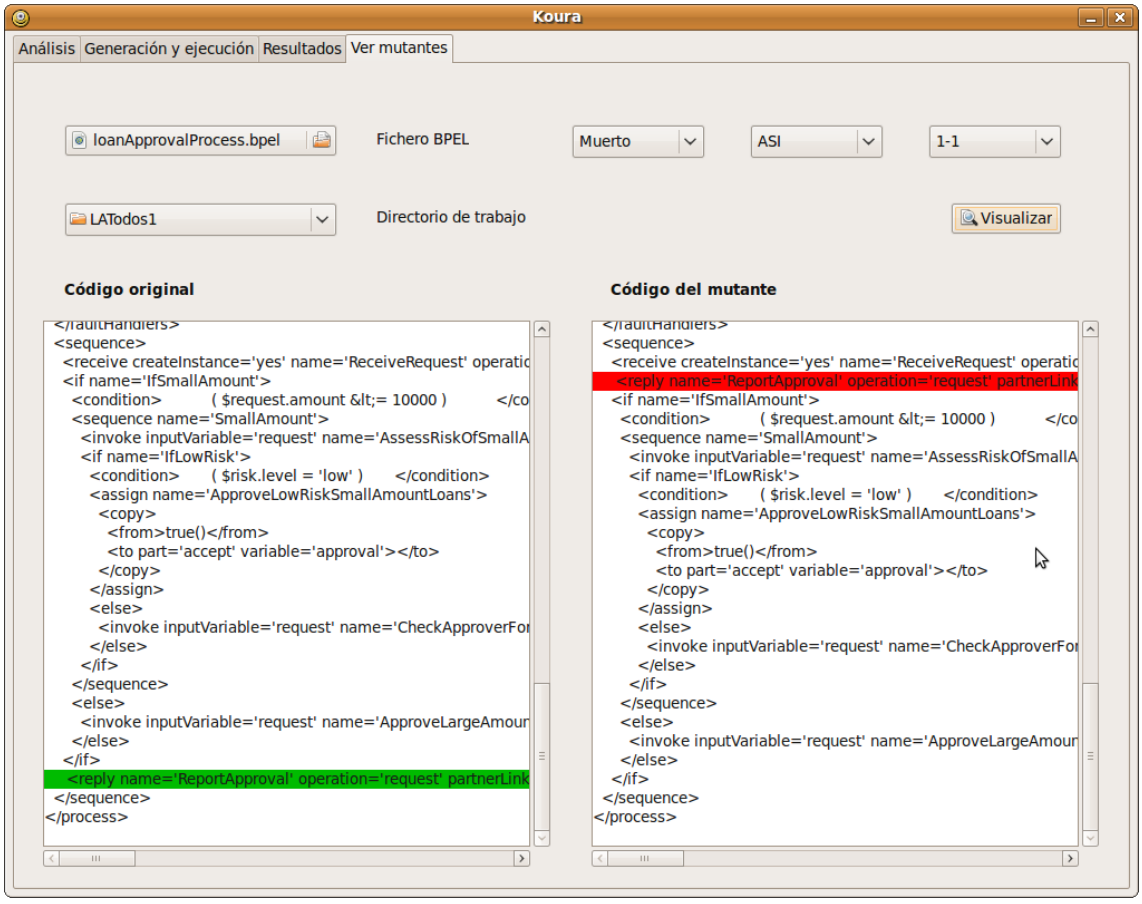

Figura 4.48: Operador ASI

Prueba APM

El operador APM, elimina un elemento *onMessage* de una actividad *pick*. Se ven afectadas todas las líneas que compongan a dicho elemento. Como podemos observar en el fichero original aparece por completo y en el mutante desaparece.

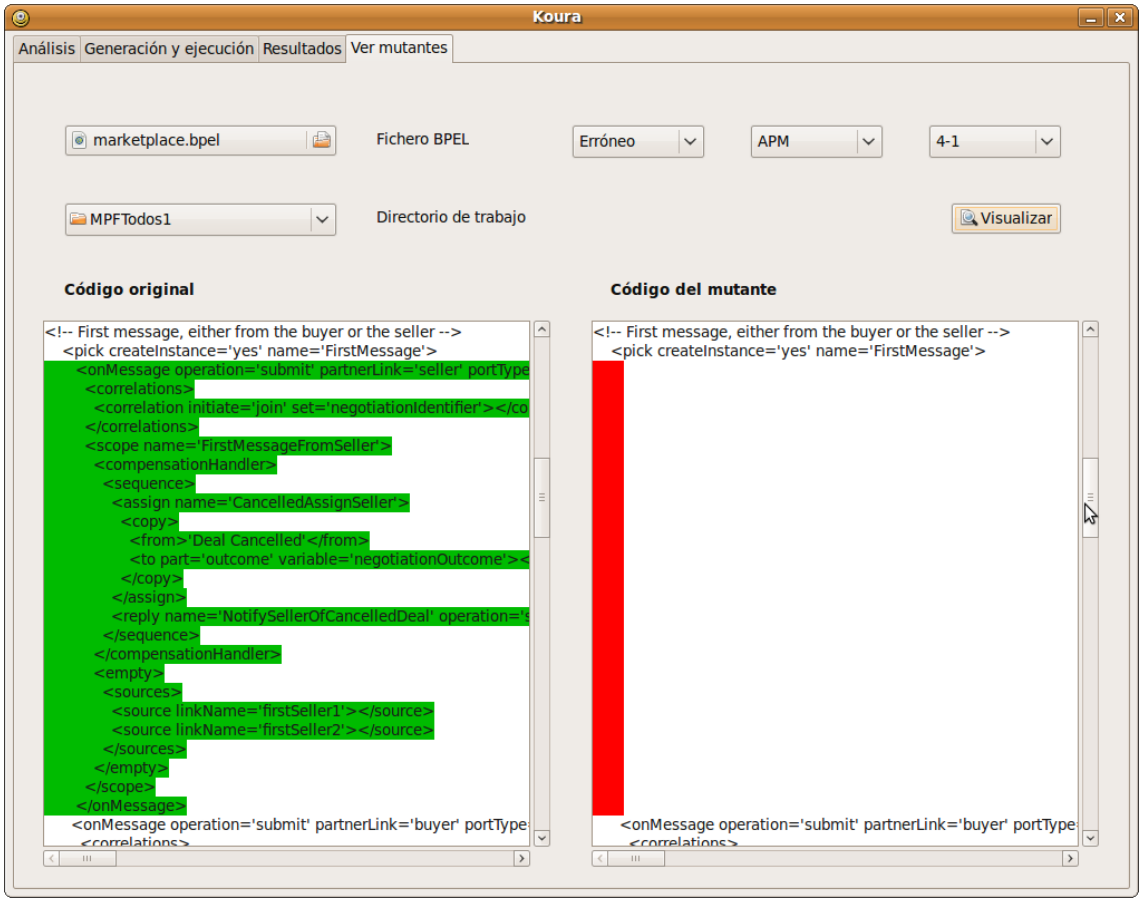

Figura 4.49: Operador APM

Prueba APA

El operador APA, elimina un elemento*onAlarm* de una actividad *pick* o de un manejador de eventos. Se ven afectadas todas las líneas que compongan al elemento onAlarm. En el fichero original aparece dicho elemento y en el mutante no.

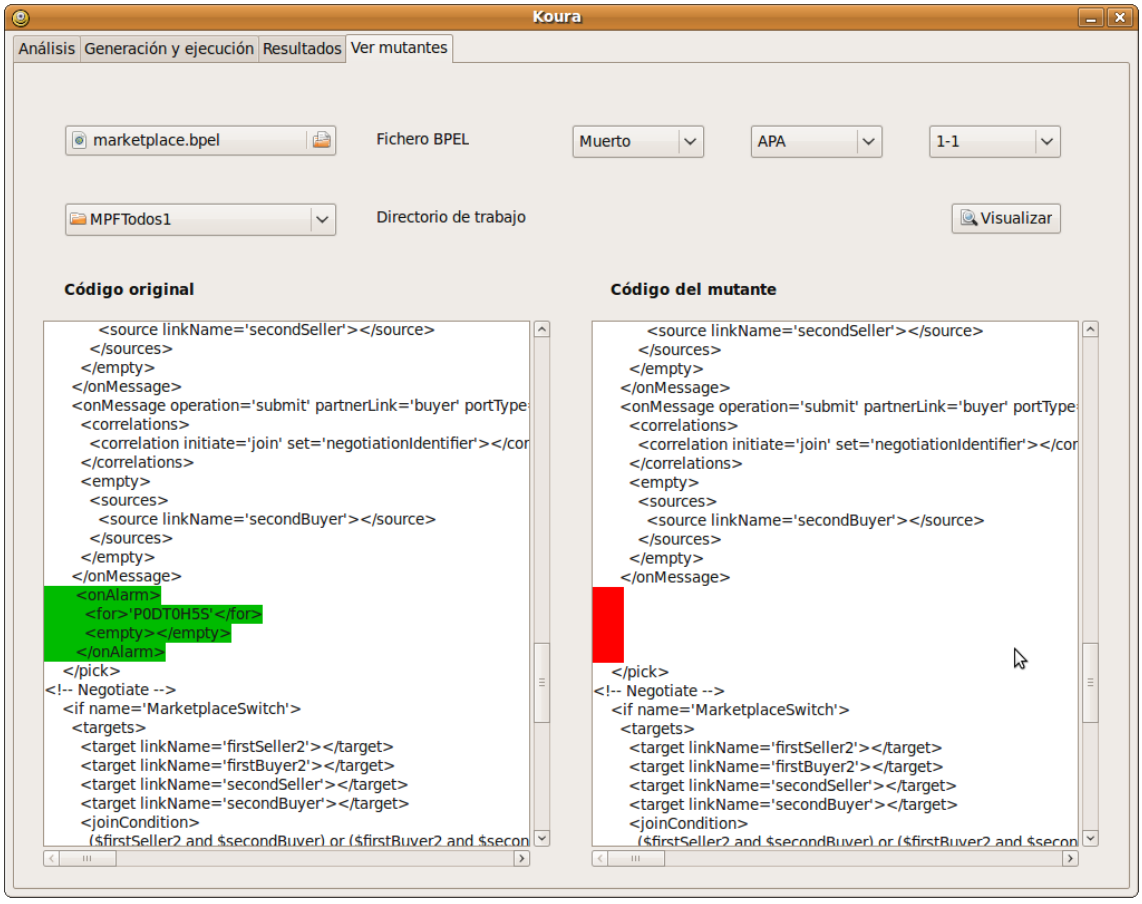

Figura 4.50: Operador APA

Prueba XMF

El operador XMF, elimina un elemento *catch* o el elemento *catchAll* de un manejador de fallos. Están afectadas todas las líneas que componen el elemento catch, en el fichero original aparece completo y en el fichero mutante no aparece.

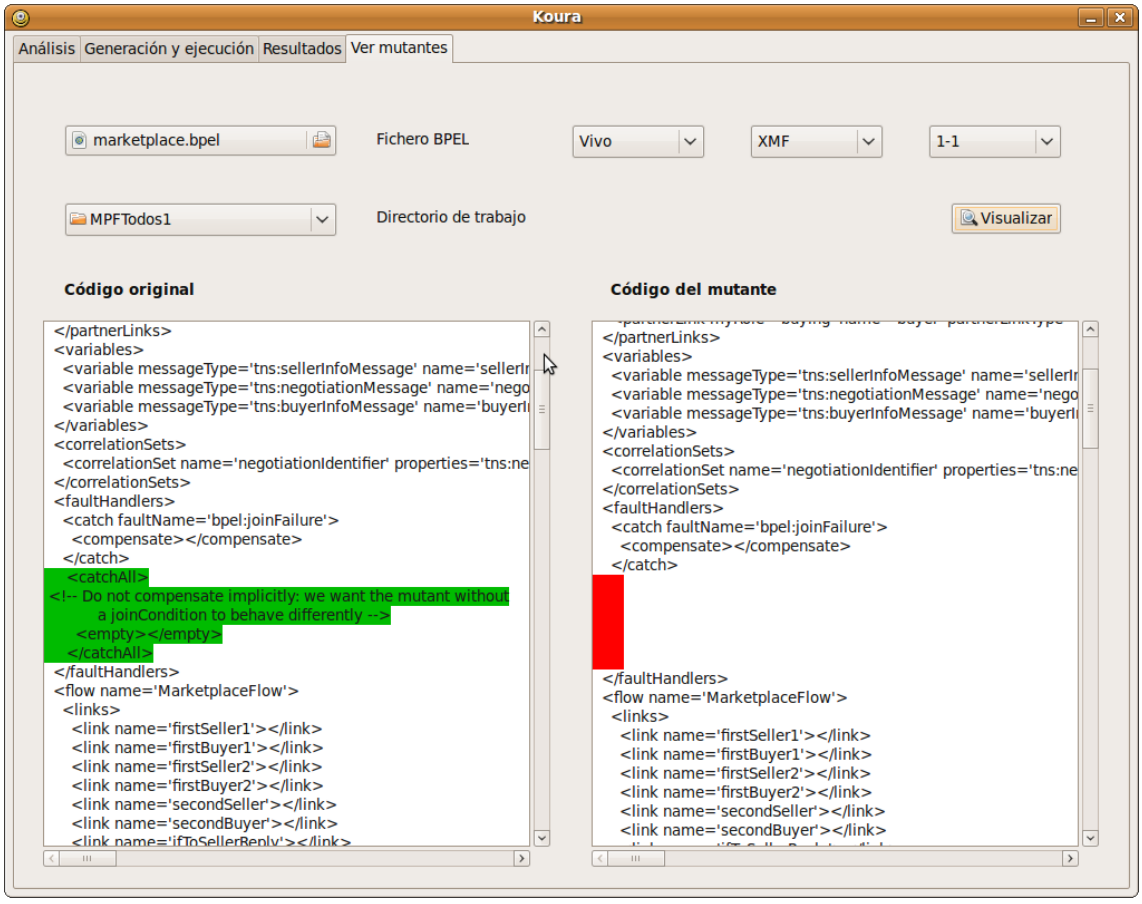

Figura 4.51: Operador XMF

Prueba XRF

El operador XRF, elimina el atributo *faultName* de una actividad *reply*. Se ve afectada una línea, la línea donde estaba el atributo de la actividad reply. Como vemos en el fichero original aparece dicho atributo y en el mutante no aparece.

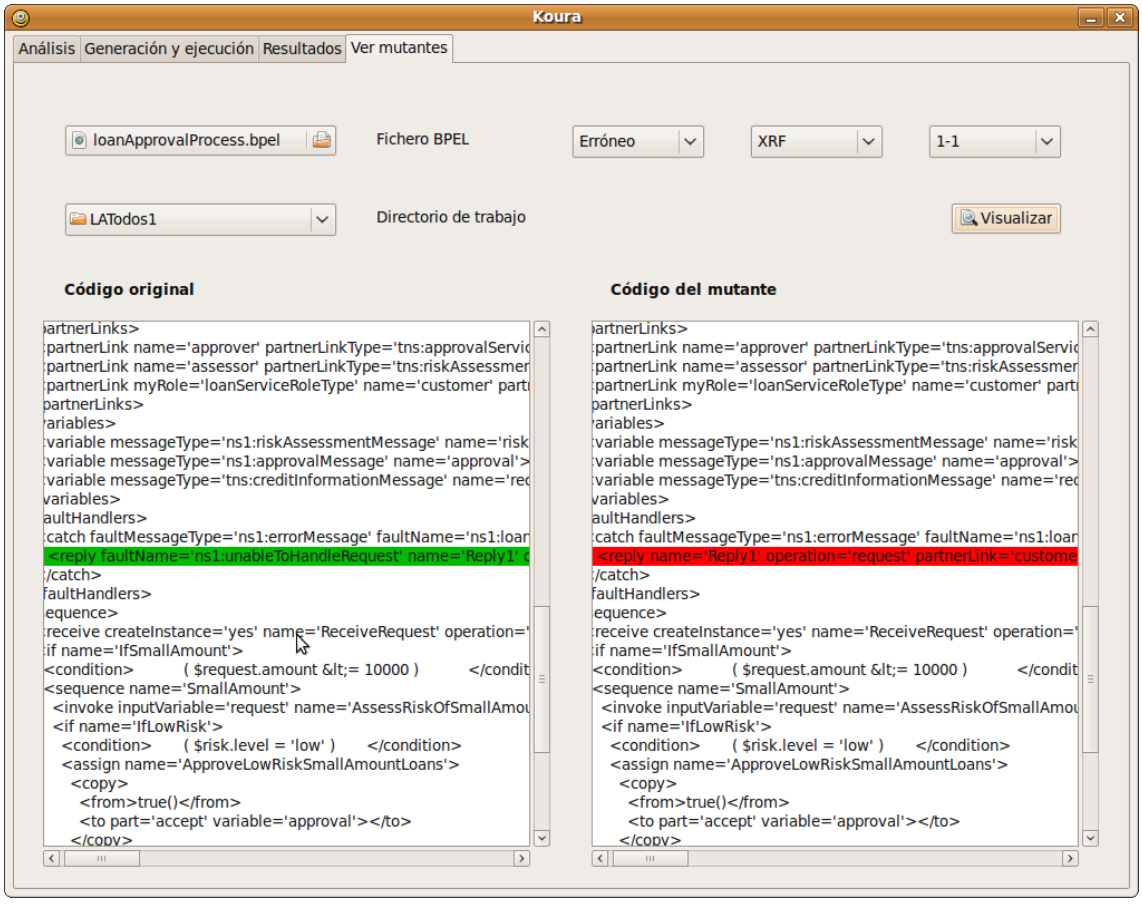

Figura 4.52: Operador XRF

Prueba XMC

El operador XMC, elimina la definición de un manejador de compensación. Se quedan afectadas todas las líneas de la definición del manejador de compensación, apareciendo totalmente en el fichero original y no apareciendo en el mutante.

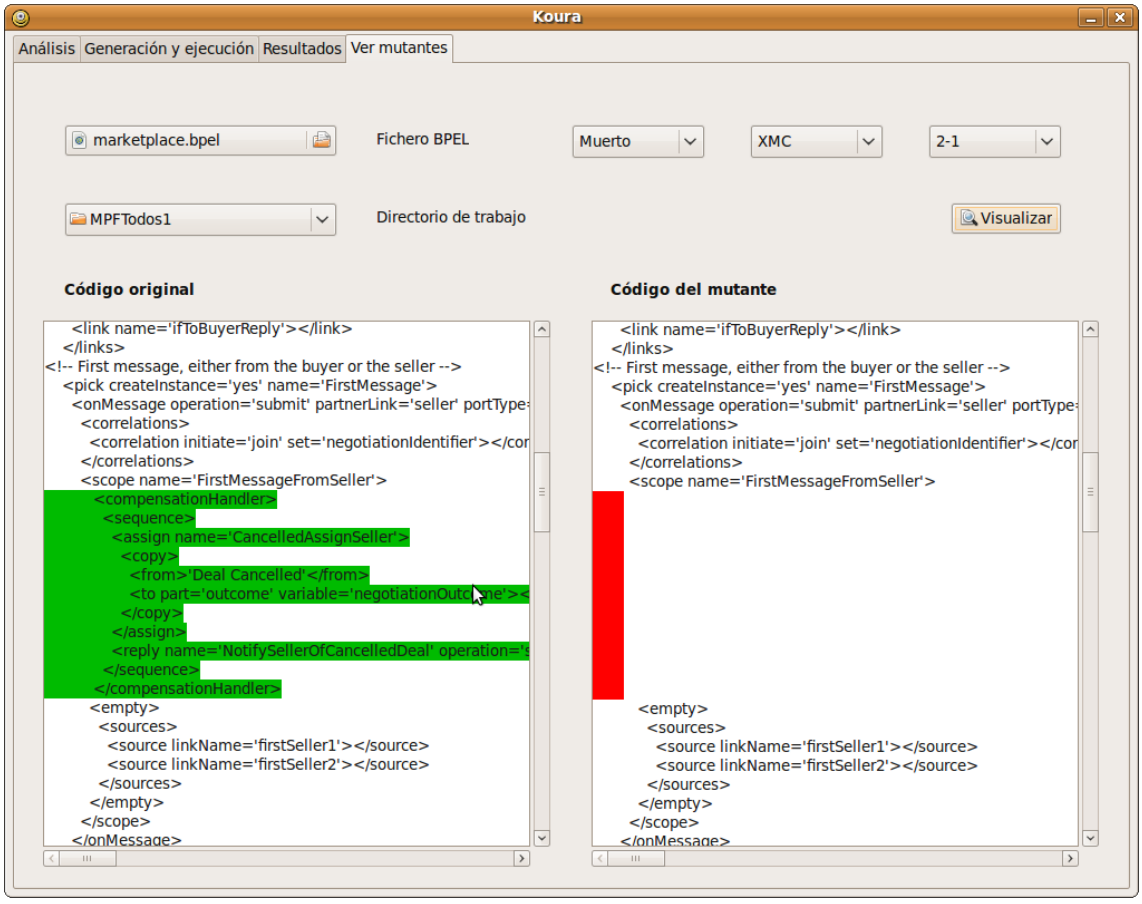

Figura 4.53: Operador XMC

Conclusiones

Tras ejecutar todos los operadores disponibles, observamos que el comportamiento que tienen es el esperado. Así que damos como válidas las funcionalidades de Koura a la hora de las comparativas entre fichero original y fichero mutante.

Capítulo 5 Resumen

Koura es un proyecto de investigación para el grupo de la Universidad de Cádiz SPI&FM que engloba varias secciones.

- El estudio y comprensión de las mutaciones y algoritmos mutantes. Definiciones, operadores que intervienen, su comportamiento, etc.
- Adaptación al algoritmo mutante, hay que estudiar la forma de trabajo, las especificaciones de sus funciones, resultados, lenguaje de programación WS-BPEL.
- Aprendizaje del nuevo lenguaje de programación a emplear para la elaboración de Koura, C#. Ventajas que nos ofrece con respecto a otros lenguajes, aplicaciones donde poder implementar Koura (monodevelop), etc.
- Aplicación del modelo en espiral, a la hora de la recolección de requisitos y de la creación del proyecto. Interpretar las necesidades de nuestro Cliente e ir mostrándole los diferentes prototipos o versiones de la aplicación a medida que vayamos avanzando.
- Estudio y adaptación de cada una de las secciones de Koura (Análisis, Generación y ejecución, Resultados y Ver mutantes), con el algoritmo genético y las necesidades del Cliente.
- Organización y recopilación de resultados para su posterior interpretación en el proyecto.
- Ampliación de los resultados obtenidos por el algoritmo genético, añadiendo a Koura la comparativa de código, la selección de mutantes según estado y un nuevo fichero resumen con resultados estadísticos más aclaratorios, entre ellos tiempo, tabla resumen de los test, tabla resumen de los operadores, etc.
- Aplicación de una guía de estilos para que el resultado visual de Koura fuese lo más ergonómico y estético posible.

Como observamos, se abarcan varias áreas de estudio y aplicación en este proyecto de investigación, en el que no sólo se ha aprendido el nuevo lenguaje de programación que se ha empleado, sino también conocimientos sobre las mutaciones, estética, funcionamiento del algoritmo genético, conexiones con otras aplicaciones y lenguajes de programación, recogida de requisitos, etc.

Si nos adentramos en las opciones de Koura se nos ofrecen las siguientes opciones para ampliar y ayudar a la aplicación GAmera:

1. Análisis
    a) Se pueden escoger todos los operadores que aparecen tras el análisis del fichero WS-BPEL
    b) Se puede escoger una selección de los operadores tras el análisis del fichero WS-BPEL
2. Generación y ejecución
    a) Todos los operadores
    b) Selección de operadores
3. Resultados
    a) Opción: Todos los operadores
        1) Fichero resumen
        2) Fichero mutantes-generados
    b) Opción: Selección de operadores
        1) Fichero resumen
        2) Fichero mutantes-generados
        3) Fichero generaciones
        4) Fichero estadísticas
4. Ver mutantes
    a) Comparativas con mutantes vivos
    b) Comparativas con mutantes muertos
    c) Comparativas con mutantes que causaron error

Tal y como se comentó antes, la aplicación entra en la presentación de las JISBD que del grupo de investigación SPI&FM de Septiembre del 2009, ya que es una aplicación que complementa, amplía y ayuda a GAmera, la aplicación con la que trabaja conjuntamente.

También comentar que se dio una conferencia durante la semana de la ciencia de la Universidad de Cádiz en la que se dio una pequeña introducción sobre la mutación, las investigaciones en MuJava y MuClipse y del proyecto Koura.

Capítulo 6 Conclusiones

Tras realizar el proyecto Koura, sacamos las siguientes conclusiones:

- Hemos obtenido conocimientos sobre las mutaciones, los algoritmos genéticos, el funcionamiento del algoritmo de mutación, el comportamiento y funcionamiento de los operadores mutantes.
- Aplicamos el método en espiral para la recogida de requisitos y la elaboración del proyecto. También hemos estudiado las aplicaciones μJava y μEclipse para captar los requisitos y observar y ampliar con posibles mejoras. Para la creación de Koura hemos ido creando versiones y prototipos que fuimos mostrando a nuestro Cliente (profesores e integrantes del grupo de investigación SPI&FM) que fueron cambiando y mejorando los requisitos según sus necesidades.
- Aprendimos los lenguajes de programación implicados en el proyecto: C♯, Perl, WS-BPEL y el lanzamiento de scripts en el sistema operativo Linux.
- Hemos aprendido sobre la estética en las aplicaciones para obtener mejores resultados en los proyectos que vayamos a realizar.

Al finalizar el proyecto, hemos aprendido conocimientos teóricos, prácticos y estéticos y hemos puesto en práctica el método en espiral, el estudio de aplicaciones y los conocimientos adquiridos con los lenguajes de programación, con la mutación, algoritmos genéticos, etc.

Koura es una aplicación que amplía, ayuda y mejora los resultados y el manejo de GAmera, por lo que es un proyecto muy completo y amplio que se acopla perfectamente a las necesidades y a las aplicaciones con las que se ejecuta conjuntamente.

Se ha implementado la aplicación de tal modo que sea fácil ampliar a la hora de añadir nuevos operadores, ya que este es uno de los propósitos perseguidos por el grupo de investigación. A la hora de la comparativa, que sería el paso más engorroso de ampliar, ya está implementado. Se ha conseguido que Koura interprete las líneas dispares entre los ficheros de tal forma que no importa el orden de las líneas en el fichero de diferencias.

El estudio del lenguaje de programación C♯, ha sido ampliar los conocimientos ya que se asemeja al lenguaje de programación C++. Especialmente se han ampliado conocimientos en el ámbito de la creación del interfaz.

Finalmente concluir diciendo que se han alcanzado todos los objetivos del proyecto, tanto la adquisición de los conocimientos teóricos, como técnicos, como el aprendizaje de las técnicas de estilo a la hora de elaborar una aplicación y el cumplir y mejorar los requisitos del Cliente.

Capítulo 7 Manual de usuario

Antes de entrar en detalle en la descripción de cada uno de los pasos para ejecutar Koura, vamos a describir el entorno de trabajo que nos ofrece esta aplicación y luego detallaremos los pasos a seguir en cada caso.

Koura está compuesta por cuatro apartados diferenciados visualmente en pestañas. La primera de ellas "Análisis" es en la que, tras seleccionar el fichero y directorio de trabajo, realiza una llamada a **mutation tools** con la instrucción **"mutation tool analyze fichero.bpel"**. En la segunda pestaña "Generación y ejecución", hay dos posibilidades: hacer unas llamadas a **mutation tools** con las instrucciones **"mutation tool apply ficherobpeloriginal OPERADOR INSTRUCCION ATRIBUTO"** para la generación de mutantes, **" mutation tool compare ficheropruebas.bpts ficherobpeloriginal bpel.out"** para la salida de cada mutante y **"mutation tool run ficheroprueba.bpts ficherobpeloriginal"** para la ejecución de los mutantes generados o llamar a **GAmera**: **gamera seed file gamera.conf**, en la que ejecutaríamos una selección de los mutantes y no la ejecución de todos como con la otra opción. En la tercera pestaña "Resultados", observamos los resultados obtenidos después de los procesos anteriores, no sólo un resumen de los valores obtenidos, sino también podemos visualizar cada uno de los ficheros que se generan según la opción seleccionada en el paso anterior. Y finalmente la pestaña "Ver mutante", que nos va a permitir comparar según el estado del mutante, el operador empleado y el test utilizado (según estos dos parámetros anteriores), las diferencias entre los mutantes y el original.

Indicar que cada una de estas pestañas, aunque siguen un proceso lineal, pueden ser abarcadas de forma independiente según las necesidades del usuario. Simplemente indicando el fichero y directorio de trabajo en la sección o pestaña actual, se podrán volver a realizar las operaciones correspondientes. Así, por ejemplo, si queremos visualizar varios "Resultados" de un mismo fichero bpel, sólo tendremos que cambiar el directorio de trabajo.

7.1. Análisis

En esta pestaña tendremos que escoger el fichero bpel con el que queremos realizar nuestro análisis y el directorio de trabajo. Si empezamos desde cero, que sería lo normal si empezamos en esta sección, tendremos que crear la carpeta que será nuestro directorio de trabajo.

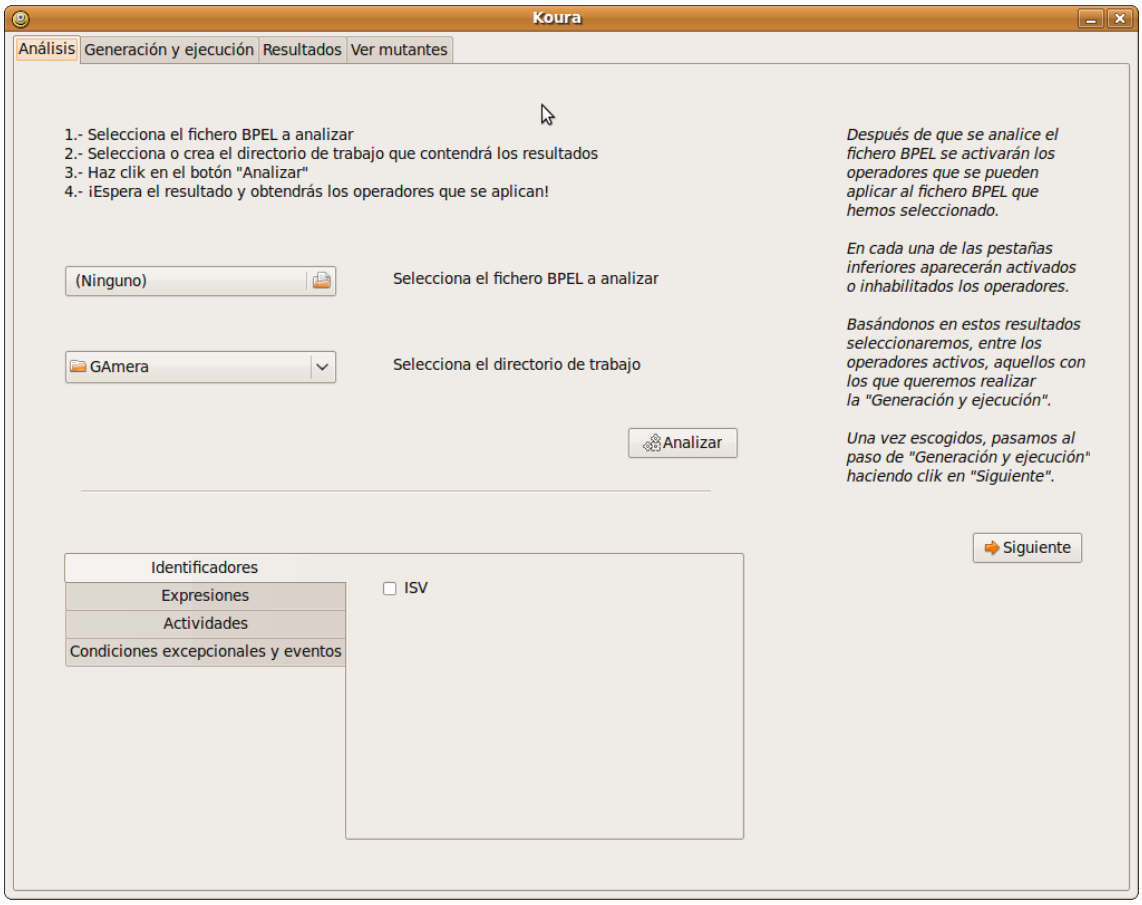

Figura 7.1: Captura Koura - Pestaña Análisis

Una vez realizado este paso haremos clik en “Aceptar” y tras realizar las operaciones correspondientes, se nos señalarán en la zona inferior cada uno de los operadores que se han empleado en dicho análisis. Los que no se emplean aparecen desactivados y los que sí están indicados con un “tick”.

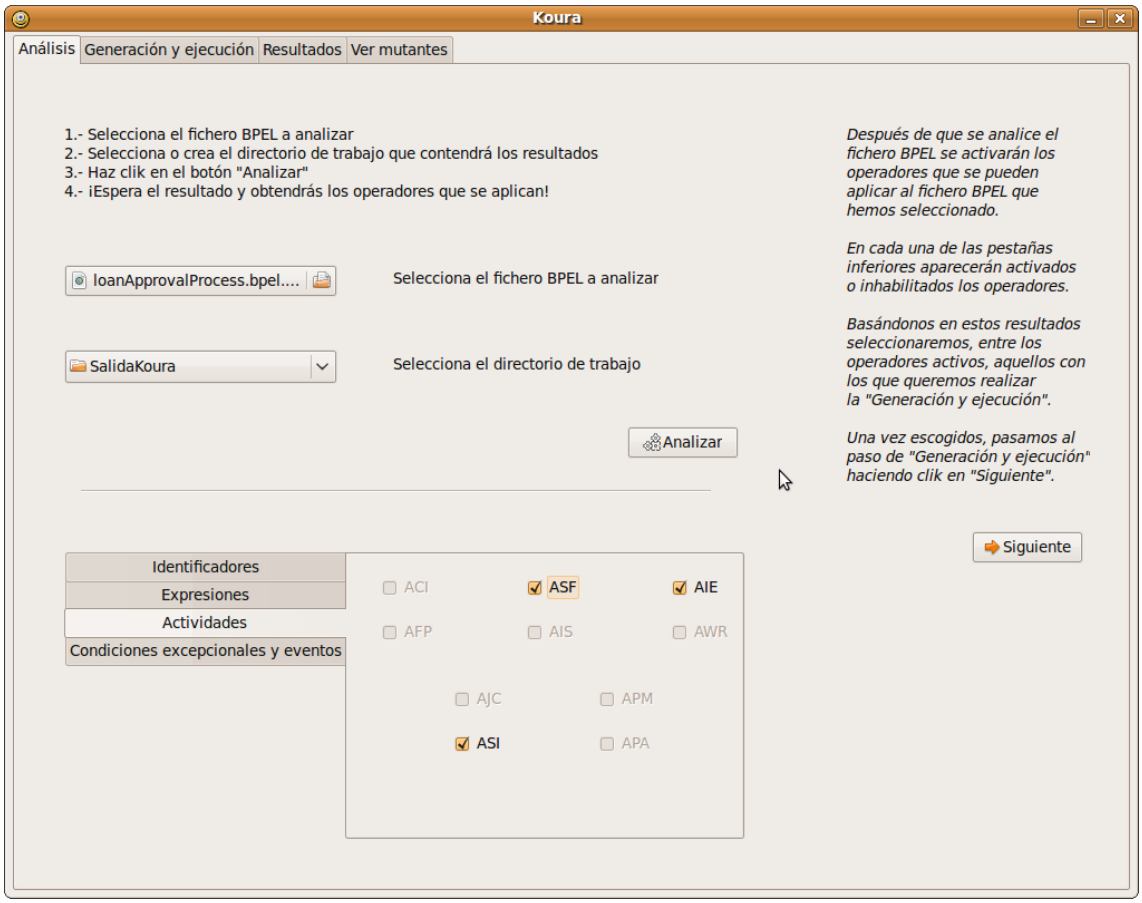

Figura 7.2: Captura Koura - Pestaña Análisis - Visualización de operadores

Podremos indicar, que a pesar de que se haya empleado en el análisis no lo vamos a querer seguir utilizando si le quitamos su selección, le quitamos el “tick”.

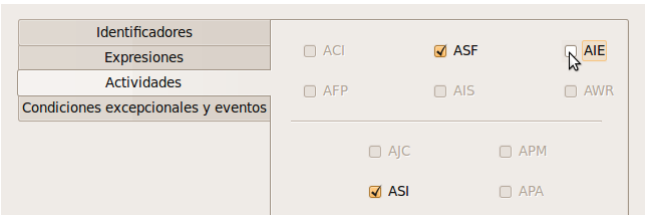

Figura 7.3: Captura Koura - Pestaña Análisis - Seleccionamos operadores

Se guarda la configuración de operadores y pasamos a “Generación y ejecución”.

7.2. Generación y ejecución

Aquí podemos observar dos partes diferenciadas: la parte superior es para la llamada a "GAmera", en la que trabajaremos con todos los mutantes, la parte inferior es para la llamada al algoritmo genético seleccionando los mutantes a trabajar.

Si venimos de la pestaña de "Análisis" el fichero y el directorio de trabajo vendrán por defecto, si queremos cambiarlos sólo tendremos que realizar los mismos pasos que dimos en la sección anterior.

7.2.1. Trabajamos con todos los mutantes

Para trabajar con todos los mutantes utilizaremos la parte superior de esta sección. El fichero y directorio de trabajo son los que vienen de la pestaña de "Análisis" luego, a no ser que queramos cambiarlos, no tenemos que modificar nada. Sólo nos quedaría seleccionar el fichero .bpts (el fichero Test), el cual nos dará el número correspondiente a los casos de test.

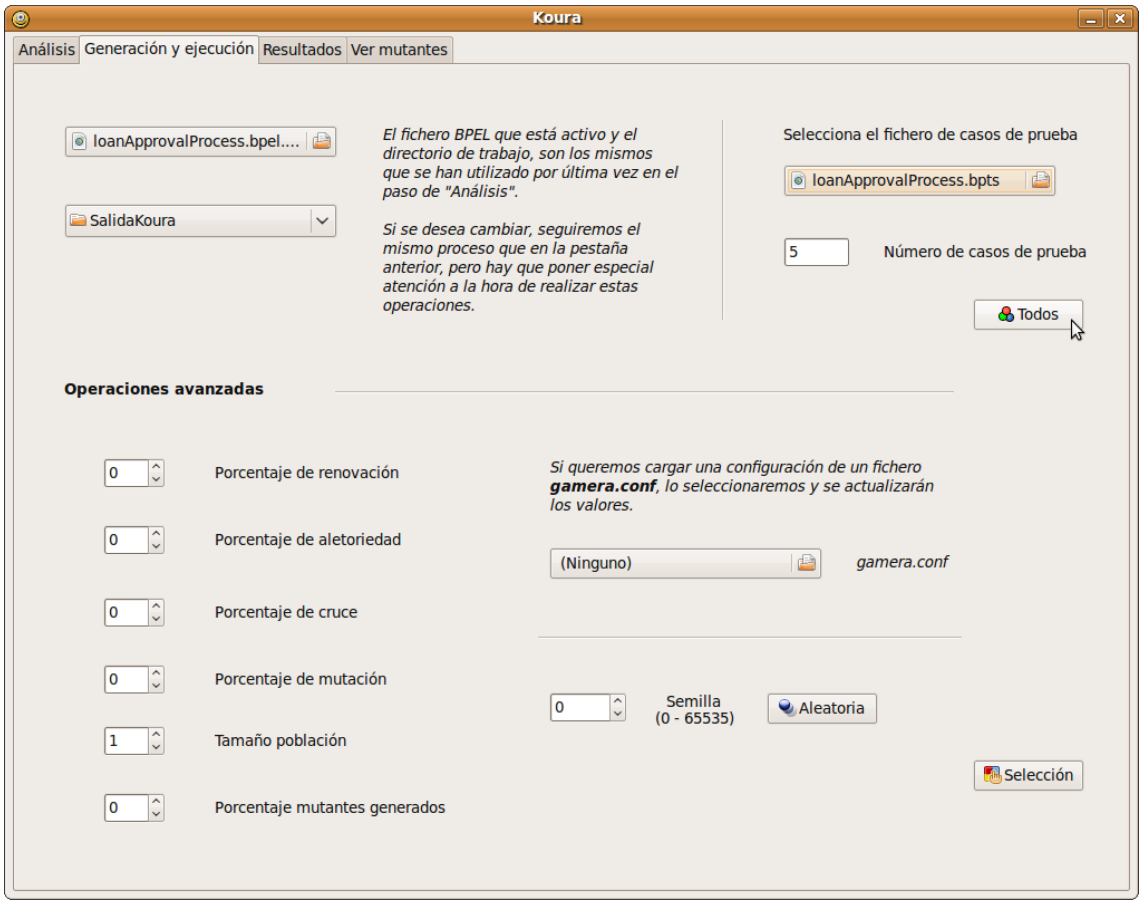

Figura 7.4: Captura Koura - Pestaña Generación y ejecución - Estudiamos todos

Una vez realizado este paso sólo tendremos que hacer clik en el botón "Todos" y una vez que se hayan realizado las operaciones correspondientes a las instrucciones **"mutation tool apply ficherobpeloriginal OPERADOR INSTRUCCION ATRIBUTO"** para la generación de mutantes, **" mutation tool compare ficheropruebas.bpts ficherobpeloriginal bpel.out"** para la salida de cada mutante y **"mutation tool run fichero-prueba.bpts ficherobpeloriginal"** para la ejecución de los mutantes generados. Directamente se nos muestra la siguiente pestaña de "Resultados".

7.2.2. Seleccionamos los mutantes con los que queremos trabajar

El trabajo con una sección de mutantes que nosotros determinamos tiene que trabajase tanto con la parte superior como la inferior de esta sección. La parte superior ya la conocemos, pasemos a la parte inferior.

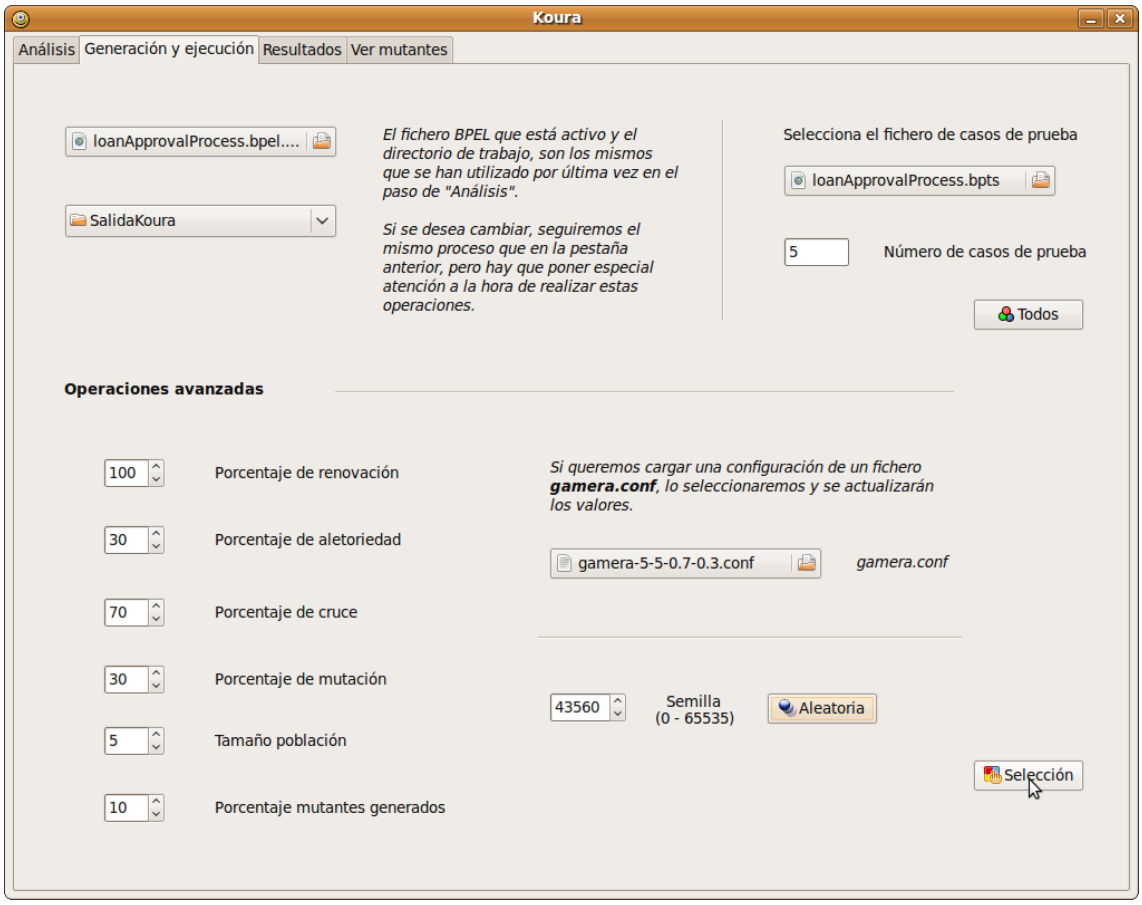

Figura 7.5: Captura Koura - Pestaña Generación y ejecución - Estudiamos selección

En ella se deben de completar los siguientes parámetros:

- Porcentaje sustituir
- Porcentaje de la probabilidad de cruce
- Porcentaje de la probabilidad de mutación
- Porcentaje nuevo
- Tamaño de la población
- Porcentaje de mutantes generados
- Semilla

Estos parámetros pueden completarse uno a uno, o bien podemos cargar la configuración desde cualquier fichero "gamera.conf" y modificar posteriormente según nuestras necesidades.

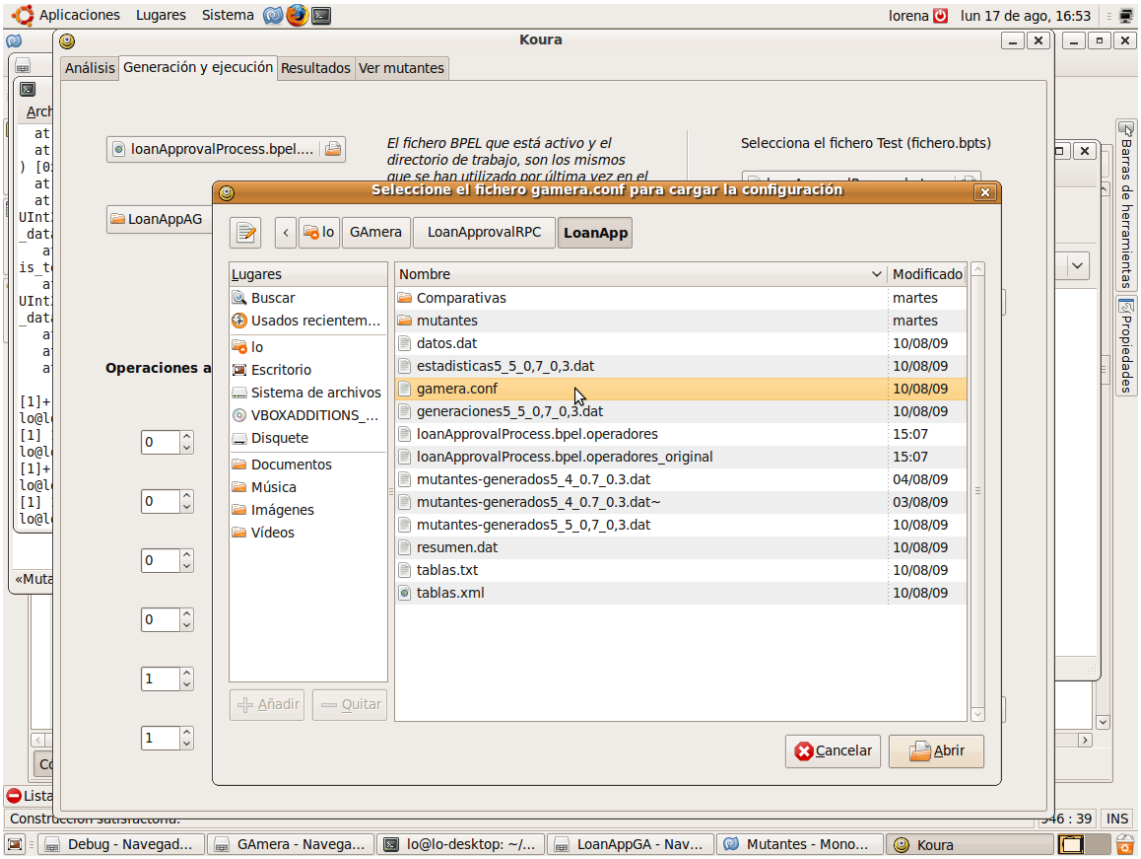

Figura 7.6: Captura Koura - Pestaña Generación y ejecución - Configuración externa

Observaciones:

1. A la hora de cargar la configuración desde un fichero "gamera.conf" se nos completará el valor para el parámetro "Número de casos de test", en el caso de que previamente no hayamos seleccionado el fichero .bpts. Si hemos seleccionado ya uno, prevalecerá el valor del fichero .bpts seleccionado y no el del fichero "gamera.conf".

2. En el caso de que empecemos directamente en esta pestaña y no tengamos hecho un análisis previo, Koura lo detectará y nos pasará a la primera sección o pestaña de "Análisis".

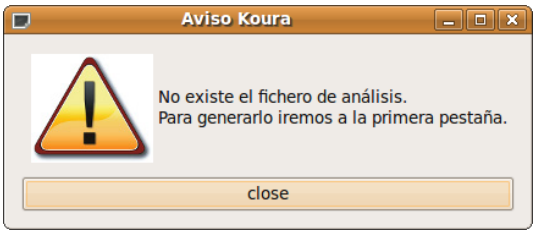

Figura 7.7: Captura Koura - Pestaña Generación y ejecución - Aviso fichero de análisis

3. Cuando se nos ha olvidado completar el campo del fichero de test (fichero.bpts), Koura nos avisa para que seleccinemos el fichero y podamos seguir trabajando.

Figura 7.8: Captura Koura - Pestaña Generación y ejecución - Aviso fichero de test fichero.bpts

7.3. Resultados

Si hemos llegado a esta pestaña tras seguir el proceso de "Análisis + Generación + Ejecución", sólo tendremos que hacer clik el botón "Calcular" y se nos mostrará en pantalla los resultados de dicho proceso. En caso contrario, tendremos que seleccionar el fichero y directorio de trabajo y pulsar el botón "Calcular", en el caso de que no encuentre los ficheros generados a la hora de realizar la "Generación y ejecución" nos devolverá a la pantalla anterior.

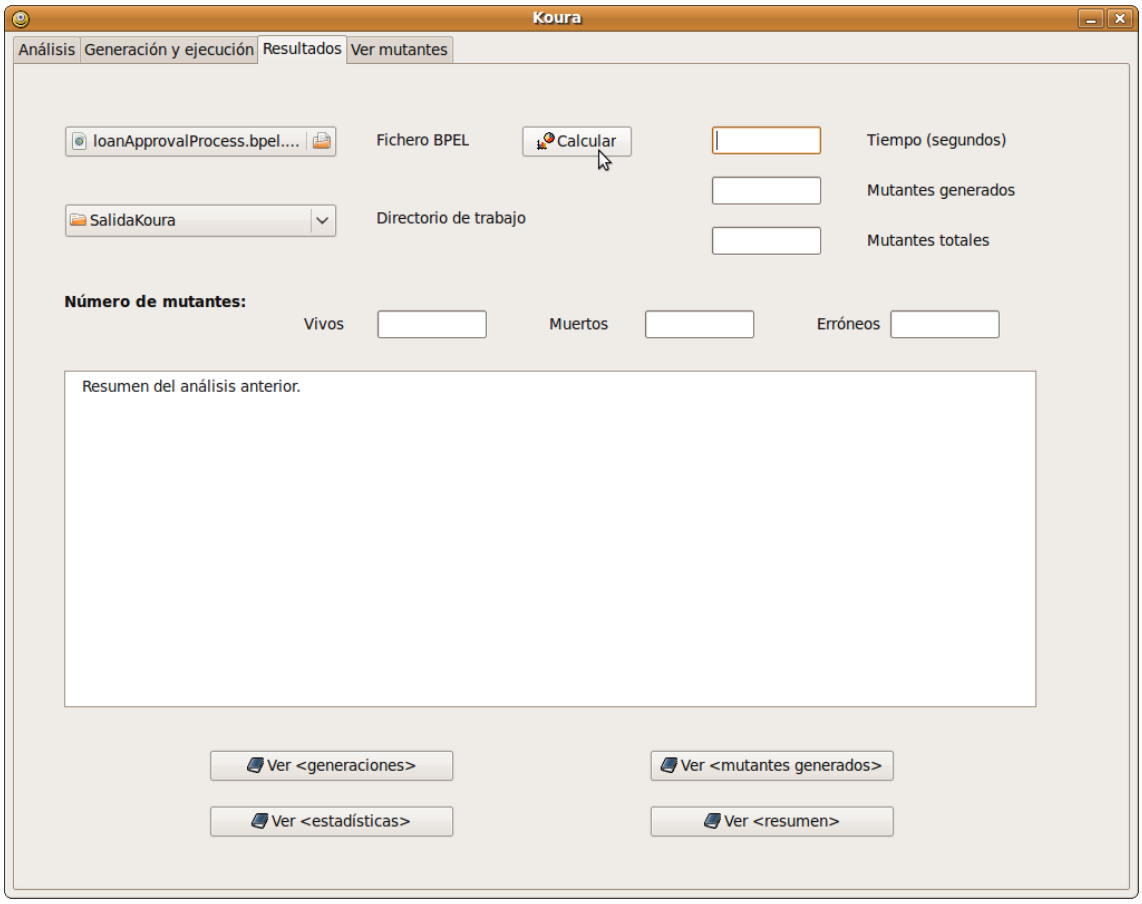

Figura 7.9: Captura Koura - Pestaña Resultados

Los resultados que se nos muestran son los siguientes:

- Tiempo: Segundos que ha tardado en realizar las operaciones.
- Número de mutantes totales
- Número de mutantes generados

- Número de mutantes
  - Vivos
  - Muertos
  - Error

A continuación aparece un resumen detallado de cada operador que se ha empleado en este proceso, indicándonos:

- Nombre
- Número de mutantes que permanecen vivos
- Número de mutantes que ha matado
- Número de mutantes generados con dicho operador
- Número de errores que se han producido

Y finalmente aparece otro resumen de cada uno de los test que se han realizado indicándonos:

- Número del test
- Número de mutantes que ha matado

Estos resultados se pueden contemplar de igual modo en el fichero *resumen.dat* que se indica en la parte inferior de la sección y en el fichero XML que hemos generado y que se encuentra dentro del directorio de trabajo y que denominamos "tablas.xml".

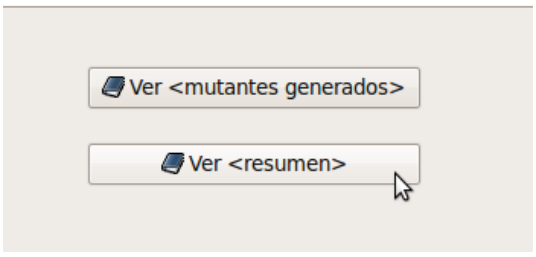

Figura 7.10: Captura Koura - Pestaña Resultados - Botón resumen

Según si hemos decidido trabajar con todos o con una selección de los operadores, tendremos la posibilidad de visualizar o no una serie de ficheros. Hablamos, en el caso de escoger la opción de trabajar con una sección de mutantes:

- *generacionesXX_XX_XX_XX.dat*

- *mutantes-generadosXX_XX_XX_XX.dat*
- *estadisticasXX_XX_XX_XX.dat*
- *resumen.dat*

En el caso de que queramos trabajar con todos, sólo se nos muestran:

- *mutantes-generadosXX_XX_XX_XX.dat*
- *resumen.dat*

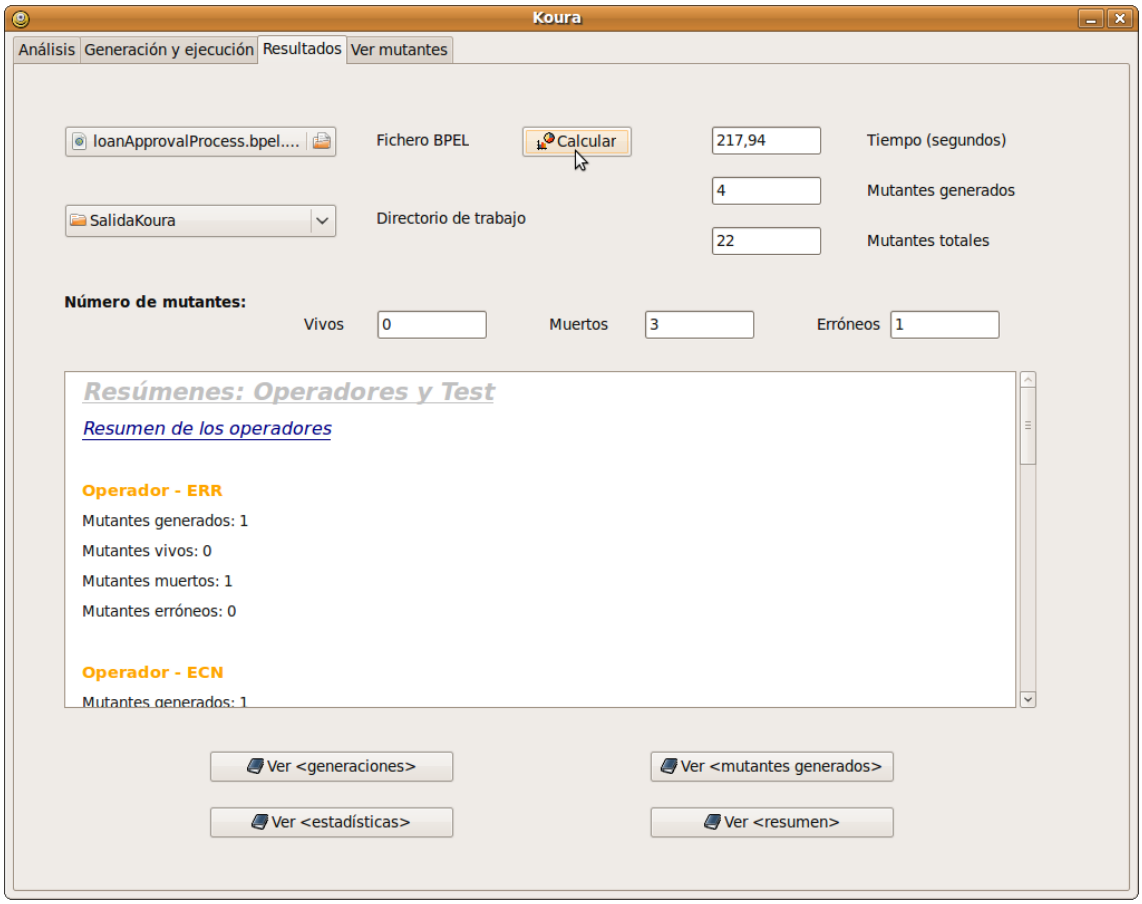

Figura 7.11: Captura Koura - Pestaña Resultados - Estudiamos una selección

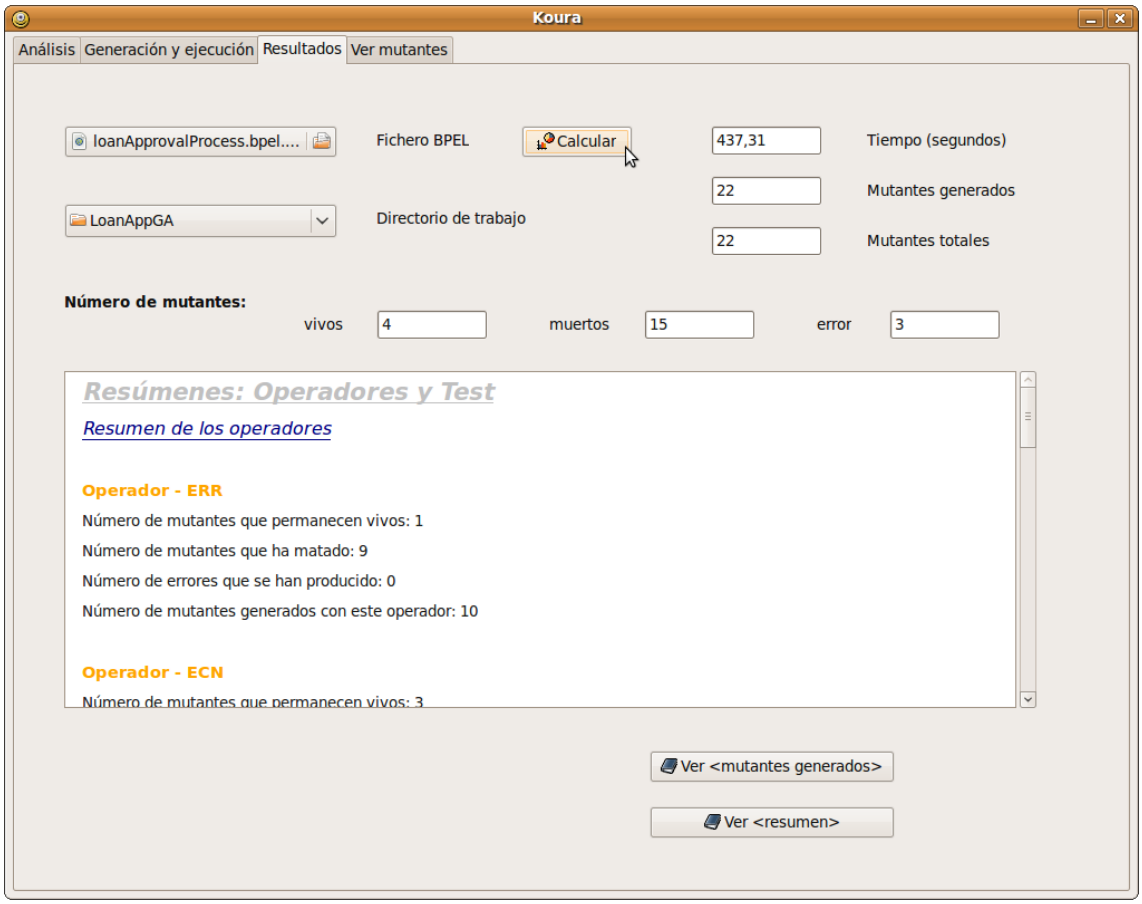

Figura 7.12: Captura Koura - Pestaña Resultados - Estudiamos todos

7.4. Ver mutante

Si hemos llegado a esta pestaña realizando todos los pasos no tendremos que seleccionar el fichero ni el directorio de trabajo, nos dirigiremos a los desplegables “Estado”, “Operador” y “Detalles”. En el primero de ellos están los tres estados en los que puede estar un mutante: Vivo, Muerto o Error. Tras escoger uno de estos estados, nos dirigimos al siguiente desplegable “Operador” y se nos aparecerá aquellos operadores que están en el estado que hemos indicado, seleccionaremos uno. Y finalmente seleccionaremos en “Detalles” la instrucción y atributo **I-A** correspondiente a dicho operador en el estado indicado. Una vez que tengamos esto hacemos clik en “Visualizar” y se nos mostrará en la región izquierda el fichero original y en la derecha el mutante, el fichero original tiene señalado en verde la diferencia y el mutante en rojo.

Dependiendo del operador que hayamos seleccionado se nos mostrará la diferencia de una forma u otra: bien se nos señala una línea, dos líneas separadas o un conjunto de líneas seguidas, esto se comprenderá mejor visualizando los diferentes operadores y entendiendo qué es lo que hace cada uno.

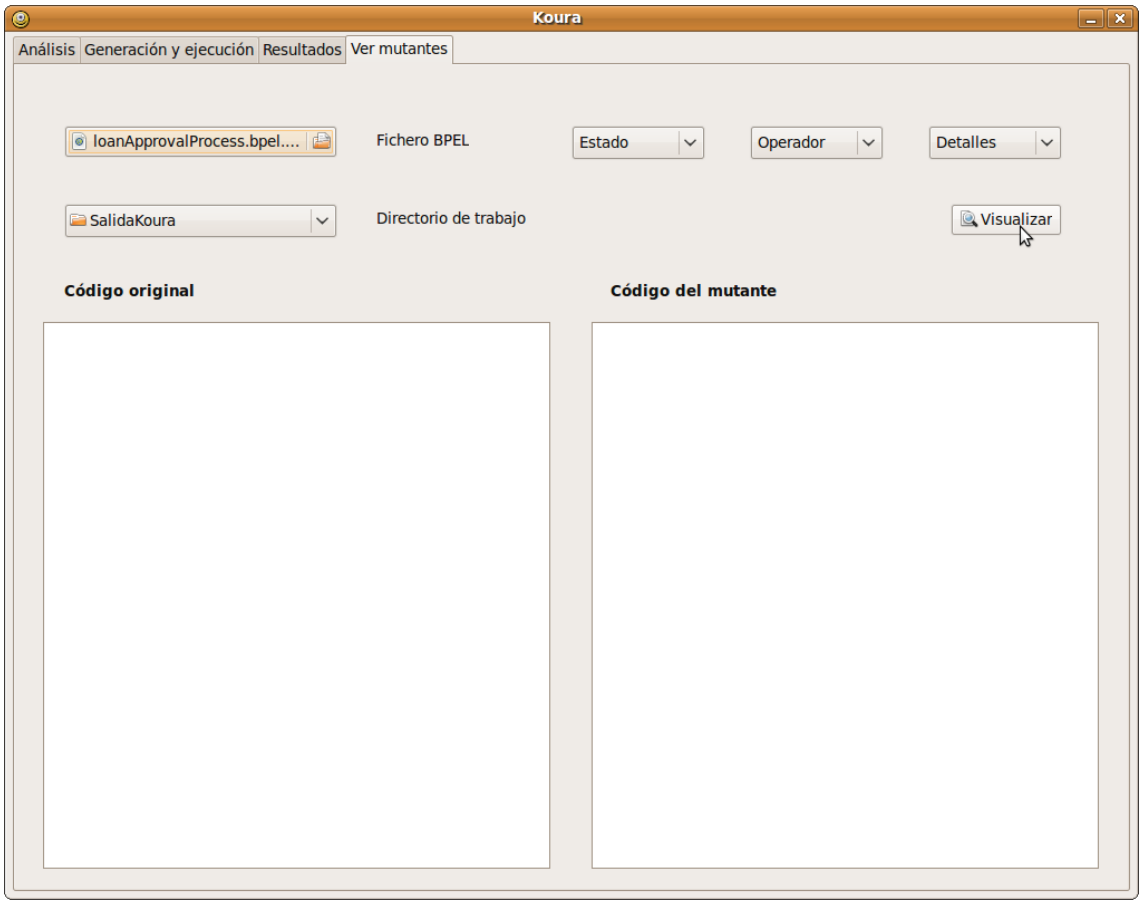

Figura 7.13: Captura Koura - Pestaña Ver mutantes

Aquí una tabla aclaratoria de los operadores, su funcionalidad y la salida que nos mostraría a la hora de visualizar las comparativas:

Operador	Descripción	Visualización en Koura
	Mutación de Identificadores	
ISV	Sustituye el identificador de una variable por el de otra del mismo tipo	Tanto en el original como en el mutante aparecerá la misma línea coloreada
	Mutación de Expresiones	
EAA	Sustituye un operador aritmético por otro del mismo tipo	Tanto en el original como en el mutante aparecerá la misma línea coloreada
EEU	Elimina el operador - unario de cualquier expresión	Idem
ERR	Sustituye un operador relacional por otro del mismo tipo	Idem
ELL	Sustituye a un operador lógico por otro del mismo tipo	Idem
ECC	Sustituye un operador de camino por otro del mismo tipo	Idem
ECN	Modifica una constante numérica	Idem
EMD	Modifica una expresión de duración	Idem
EMF	Modifica una expresión de fecha límite	(*)

(*) De estos operadores todavía no disponemos de ejemplos para aplicarlos y ver las diferencias con el fichero original. Pero el comportamiento de éstos será semejante a los que tienen una descripción similar. Para aquellos que no comparten una descripción, habría que estudiar el ejemplo resultante. Para Koura no hay problemas a la hora de realizar la comparativa y visualizarlos, ya que está implementada de tal forma que irá comparando y coloreando todas aquellas diferencias sin importarle el orden en que se encuentren.

Operador	Descripción	Visualización en Koura
	Mutación de Actividades	
	Relacionados con la concurrencia	
ACI	Cambia el atributo *createInstance* de las actividades de recepción de mensajes a *no*	Tanto en el original como en el mutante aparecerá la misma línea coloreada
AFP	Cambia una actividad *forEach* secuencial a paralela	Tanto en el original como en el mutante aparecerán las dos mismas líneas coloreadas (apertura y cierre)
ASF	Cambia una actividad *secuence* por una actividad *flow*	Idem
AIS	Cambia el atributo *isolated* de un *scope* a *no*	Tanto en el original como en el mutante aparecerá la misma línea coloreada
	Mutación de Actividades	
	No concurrentes	
AIE	Elimina un elemento *elseif* o el elemento *else* de una actividad *if*	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
AWR	Cambia una actividad *while* por una *repeatUntil* y viceversa	Tanto en el original como en el mutante aparecerá la misma línea coloreada
AJC	Elimina el atributo *joinCondition* de cualquier actividad en la que aparezca	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
ASI	Intercambia el orden de dos actividades hijas de una actividad *sequence*	Aparecerá coloreada una línea, pero en el original aparecerá en una posición diferente a la del mutante
APM	Elimina un elemento *onMessage* de una actividad *pick*	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
APA	Elimina el elemento *onAlarm* de una actividad *pick* o de un manejador de eventos	Tanto en el original como en el mutante aparecerá la misma línea coloreada

Operador	Descripción	Visualización en Koura
	Mutación de Condiciones Excepcionales y Eventos	
XMF	Elimina un elemento *catch* o el elemento *catchAll* de un manejador de fallos	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
XRF	Elimina el atributo *faultName* de una actividad *reply*	Tanto en el original como en el mutante aparecerá la misma línea coloreada
XMC	Elimina la definición de un manejador de compensación	Aparecerá coloreado en el original las líneas implicadas y en el mutante un espacio en blanco donde deberían de aparecer
XMT	Elimina la definición de un manejador de terminación	(*)
XTF	Cambia el fallo lanzado por una actividad *throw*	(*)
XER	Elimina una actividad *rethrow*	(*)
XEE	Elimina un elemento *onEvent* de un manejador de eventos	(*)

Veamos gráficamente lo que aclaramos en la tabla anterior:

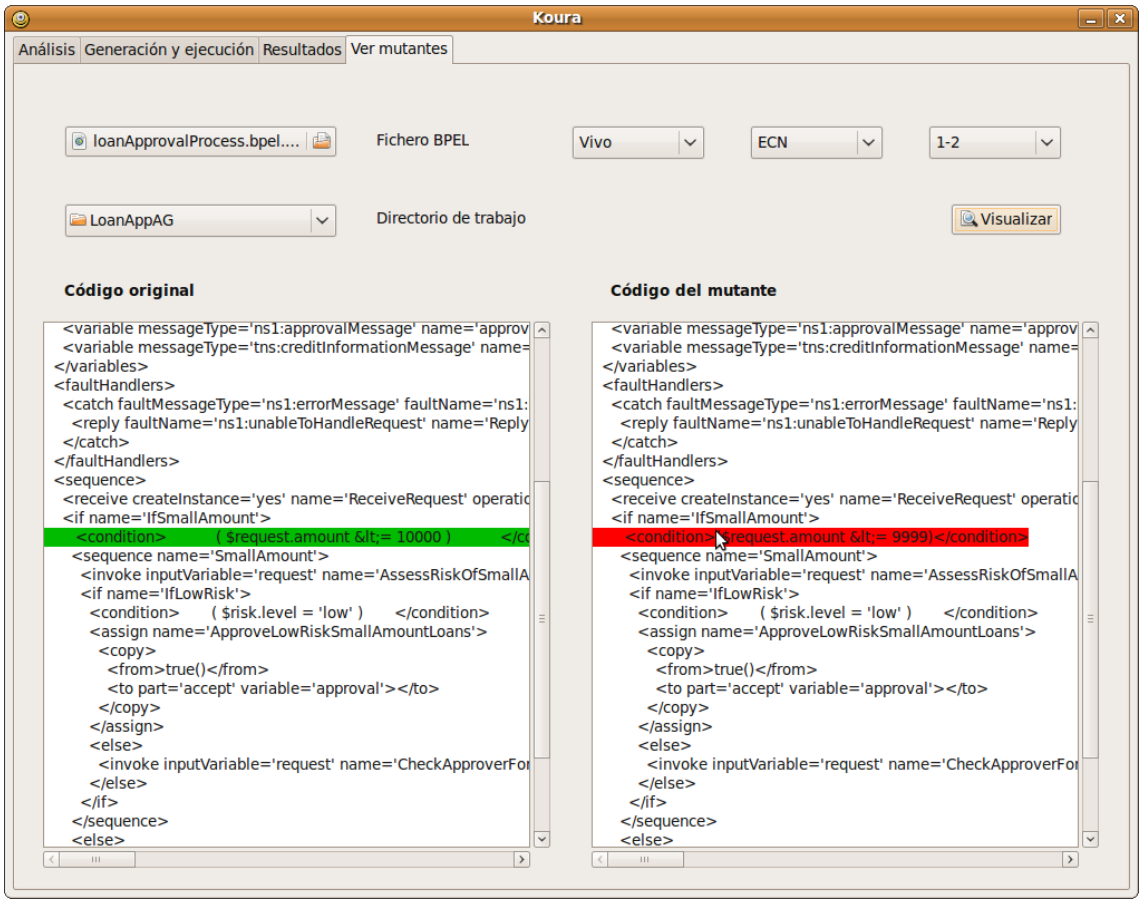

Figura 7.14: Captura Koura - Pestaña Ver mutantes - Ejemplo comparativa

A medida que vayamos cambiando el estado se irá actualizando el desplegable de los operadores y de igual modo se irá actualizando el desplegable "Detalles" según el operador escogido. En el caso de que no exista ningún operador según el estado que hayamos escogido, el desplegable aparecerá desactivado y si seleccionamos en el desplegable "Estado" la opción "Estado", se nos limpiará la pantalla y los desplegables "Operador" y "Detalles".

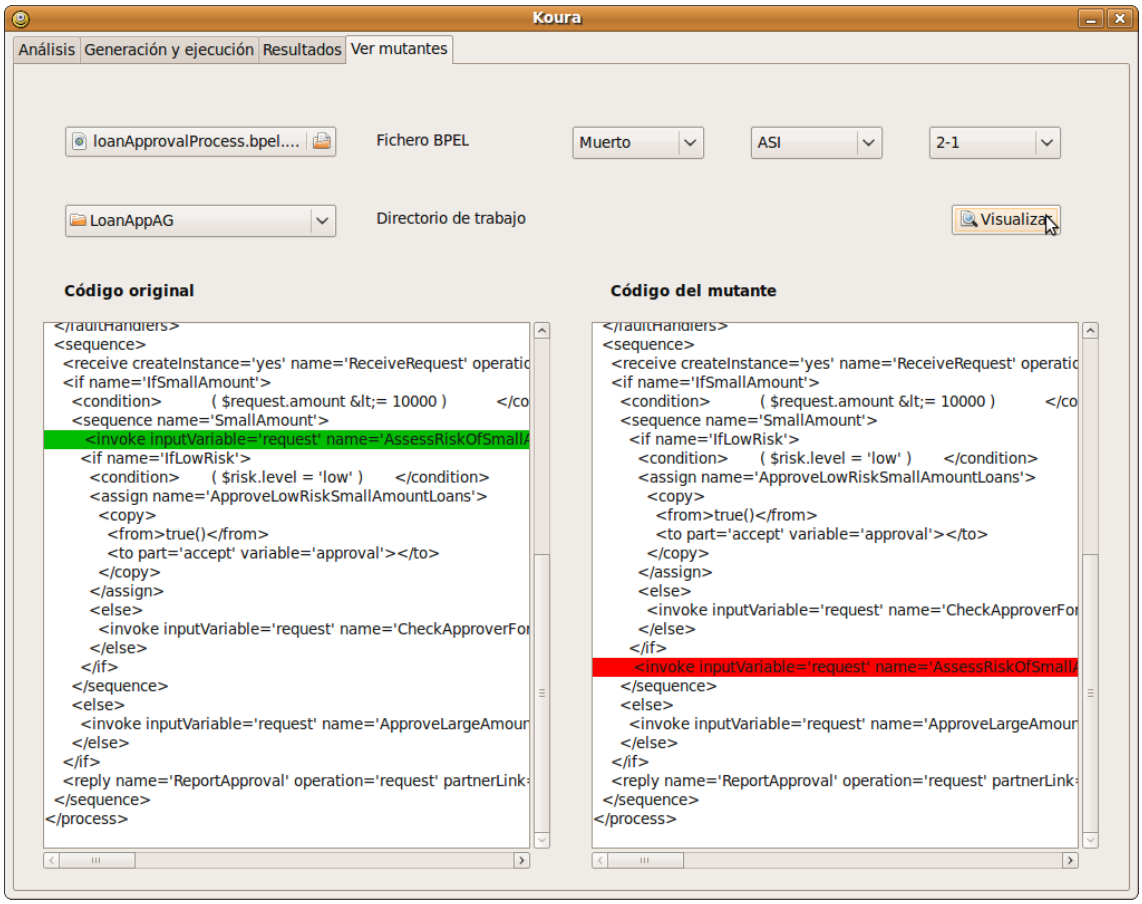

Figura 7.15: Captura Koura - Pestaña Ver mutantes - Ejemplo comparativa

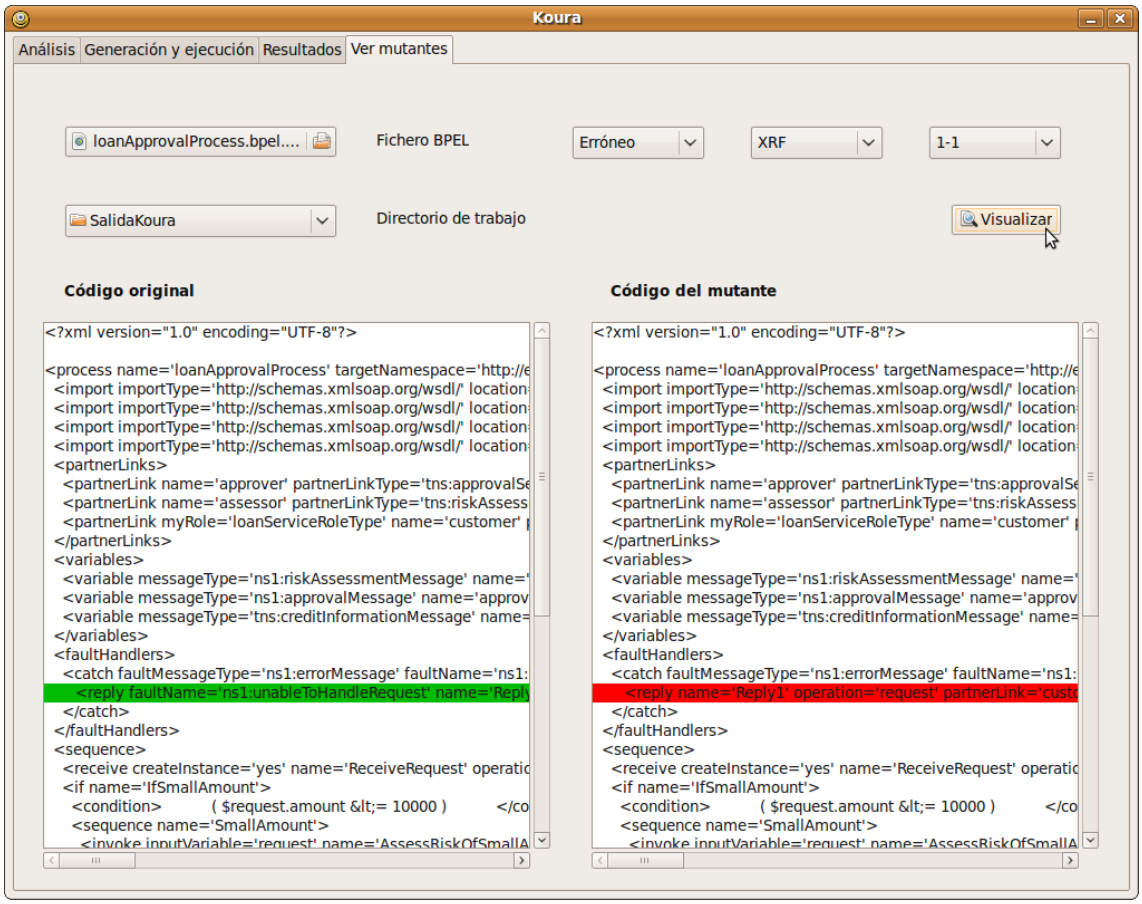

Figura 7.16: Captura Koura - Pestaña Ver mutantes - Ejemplo comparativa

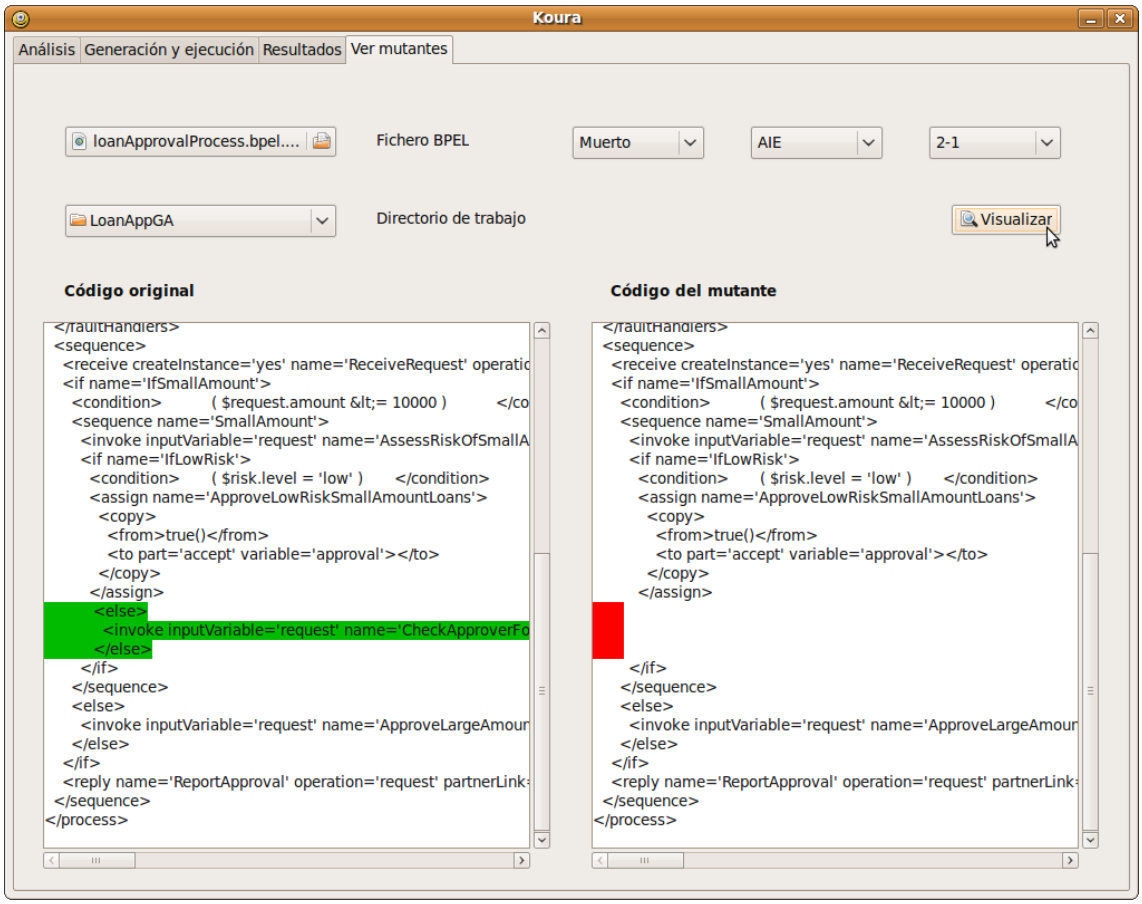

Figura 7.17: Captura Koura - Pestaña Ver mutantes - Ejemplo comparativa

Observaciones:

1. Tanto en la pestaña o sección de “Resultados” como en la de “Ver mutante”, como se supone que venimos de realizar el estudio previo de “Análisis + Generación + Ejecución”, si no encuentra un fichero denominado “datos.dat” que contiene la información necesaria para leer los resultados o mostrar las comparativas, supondrá que no hemos hecho el estudio previo y nos avisará mediante una advertencia que lo hagamos.

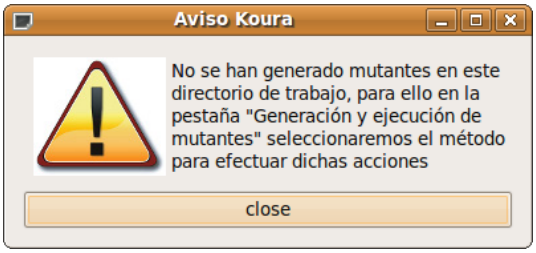

Figura 7.18: Captura Koura - Pestaña Ver mutante - Aviso no se ha hecho estudio previo

Aquí un ejemplo del contenido del fichero “datos.dat”

```
1 AG
2 valores
3 5_5_0,7_0,3
4 tiempo
5 257,249846999999
```

Listado 7.1: datos.dat

Capítulo 8 Manual de Instalación

Para poder ejecutar Koura, tendremos que instalar inicialmente GAmera ya que Koura se ejecuta sobre ésta, le ayuda, complementa y amplía. Así que antes de profundizar en la instalación de Koura, procedamos a instalar GAmera.

Los pasos de instalación de GAmera son los que vienen publicados en la Web del grupo de investigación de la Universidad de Cádiz SPI&FM. [29]

Después de la instalación de GAmera, hablaremos de los pasos a seguir para la instalación de Koura, dependencias, versiones y paquetes que necesitaremos.

8.1. Instalación de GAmera

Cada una de estas subsecciones es una tarea de alto nivel a seguir en la instalación, que luego descomponemos en tareas más simples.

8.1.1. Preparación de .bashrc

Hay que crear un directorio bin dentro de $HOME si no lo hay, y ponerlo bajo el PATH:

```
mkdir -p ~/bin
echo 'export PATH="$PATH:~/bin"' >> ~/.bashrc
```

Hemos de volver a cargar el .bashrc, o abrir y cerrar la terminal:

```
source ~/.bashrc
```

8.1.2. Instalación de Java 5

Ha de ser un entorno Sun Java 5. Casi todo funciona con OpenJDK 6, pero al ejecutar GAmera se producen unos fallos que no ocurren al usar Sun Java 5, por lo que por lo pronto lo dejaremos así. Obtener un JRE de Java 5 en GNU/Linux varía según la distribución.

En openSUSE, sería:

1. Desinstalar Java 6 (si lo tenemos):

```
sudo zypper remove java-1_6_0-sun
```

2. Instalar Java 5:

```
sudo zypper install java-1_5_0-sun
```

En una Debian/Ubuntu:

1. Instalar Java 5:

```
sudo aptitude install sun-java5-jre
```

2. Seleccionar Java 5 como JRE por defecto:

```
sudo update-alternatives-java --set java-1.5.0-sun
```

En ambos casos, tendríamos que asegurarnos de que todo está correctamente instalado y configurado. Para ello, seguiremos estos pasos:

1. Cerrar y abrir la terminal en que nos encontremos, para que se vuelvan a cargar los ficheros del perfil.
2. Comprobar que está disponible la versión correcta con:

```
java -version
```

3. Comprobar que JAVA_HOME está correctamente configurado. Al ejecutar esta orden, tendría que aparecer un único fichero, con los permisos de ejecución habilitados:

```
ls -l $JAVA_HOME/bin/java
```

8.1.3. Instalación de curl

Nuestro guión de gestión de ActiveBPEL usa los servicios web de administración de ActiveBPEL, y para ello utiliza la herramienta curl. Viene instalado por omisión en openSUSE y Ubuntu, pero si no estuviera, sería tan sencillo como ejecutar esto en Debian/Ubuntu:

```
sudo aptitude install curl
```

En openSUSE:

```
sudo zypper install curl
```

8.1.4. Descarga de los ficheros necesarios desde SVN

Necesitaremos obtener bastantes ficheros de SVN a lo largo de este tutorial. Lo más cómodo es copiar y pegar la siguiente orden en una terminal, que las descargará en un subdirectorio instalacion-gamera bajo el directorio actual. Es posible que tengamos que introducir nuestros credenciales de Redmine si es la primera vez que accedemos al SVN de Neptuno desde nuestra máquina.

```
mkdir -p instalacion-gamera
pushd instalacion-gamera
for i in deps/{activebpel-4.1-bin.tar.gz,apache-tomcat-5.5.26.zip} \
         deps/{bpelunit-CVS-200709+parches.tar.bz2,galib247.tgz} \
         src/GAmera bin/{mutationtool,mutation_analysis.jar} \
         scripts/ActiveBPEL.sh src/MutationAnalysis/test_processes
do
  svn export "http://neptuno.uca.es/svn/sources-fm/$i"
done
popd
```

Otra opción es hacer una copia de trabajo de todo el repositorio, pero esto lleva más tiempo y ocupa mucho más:

```
svn co http://neptuno.uca.es/svn/sources-fm/
```

8.1.5. Instalación de Tomcat 5.5 y ActiveBPEL 4.1

Hay que extraer (preservando la estructura de directorios) apache-tomcat-5.5.26.zip y activebpel-4.1-bin.tar.gz a /bin.

Además, hay que modificar el /.bashrc ya que $CATALINA_HOME tiene que apuntar a donde está Tomcat:

```
echo "export CATALINA_HOME='ls -d ~/bin/apache-tomcat-5.5.26'" >> ~/.
    bashrc
```

Si esta orden da fallo, es que no se ha descomprimido Tomcat 5.5 al sitio correcto. Tras cargar de nuevo .bashrc como en el apartado anterior, esto debería mostrar una lista de ficheros:

```
ls "$CATALINA_HOME"
```

Ahora nos introduciremos en /bin/activebpel-4.1 y ejecutaremos:

```
sh install.sh
```

Copiaremos el guión ActiveBPEL.sh a /bin, le daremos permisos de ejecución y lo iniciaremos con:

```
ActiveBPEL.sh start
```

Tras unos 10-15 segundos, el guión debería terminar su ejecución con éxito. Ello indica que Tomcat y ActiveBPEL están activos y funcionando correctamente. Se puede comprobar a mano desde "esta dirección":http://localhost:8080/BpelAdmin/.

Otras subórdenes útiles del guión ActiveBPEL.sh son:

- pause pausa ActiveBPEL (no mata ningún proceso)
- stop echa abajo Tomcat y ActiveBPEL (mata todos los procesos)
- restart para e inicia todo (stop y luego start)
- ls lista los PID de los procesos en ejecución
- kill PID mata al proceso BPEL con identificador PID
- killall mata todos los procesos en ejecución. Es mucho más eficiente que restart, ya que no echa abajo ActiveBPEL y Tomcat.

8.1.6. Instalar BPELUnit

Extraemos bpelunit-CVS-200709+parches.tar.bz2 en /bin, y retocamos el fichero /bin/bpelunit/conf/configuration.xml para que la línea con ActiveBPELDeployment-Directory tenga el siguiente aspecto:

```
<property name="ActiveBPELDeploymentDirectory">/home/nombre-usuario/bin/
    apache-tomcat-5.5.26/bpr</property>
```

Hemos de añadir una variable más a /.bashrc:

```
echo "export BPELUNIT_HOME='ls -d ~/bin/bpelunit'" >> ~/.bashrc
```

Ahora cargamos de nuevo /.bashrc y comprobamos que todo va bien:

```
ls $BPELUNIT_HOME
```

8.1.7. Instalar el gestor de mutantes WS-BPEL de GAmera

Este es el componente específico para el manejo de mutantes WS-BPEL en GAmera. Sólo hemos de copiar mutation_analysis.jar y mutationtool de instalacion-gamera a /bin, darle permiso de ejecución a mutationtool y ejecutar:

```
mutationtool
```

Debería dar una lista de las distintas órdenes que podemos utilizar. Podemos comprobar que funciona todo lo que tenemos hasta ahora yendo a instalacion-gamera/test_processes, donde encontraremos una serie de ejemplos con los que se ha depurado mutationtool. Iremos al directorio del ejemplo loanStructured y ejecutaremos:

```
mutationtool analyze loanStructuredNonRPC.bpel
```

Eso debería dar una lista con el número de operandos y valor máximo del atributo para cada operador de mutación. Esta otra orden ejecutará el proceso WS-BPEL original bajo un BPTS que define el conjunto de casos de prueba:

```
mutationtool run ServicioPrestamosTest.bpts loanStructuredNonRPC.bpel > salida.xml
```

Si se examina bajo Firefox, debería tener un atributo con el valor "PASSED" en el elemento raíz. Ello quiere decir que todas las pruebas han sido superadas (como debería ser). Si dice "ERROR" o "ABORT", entonces hay algún problema con nuestra instalación, o hemos olvidado iniciar ActiveBPEL.

8.1.8. Instalar el núcleo de GAmera

Primero, hay que instalar la biblioteca galib utilizada por GAmera. Para ello, habrá que extraer galib247.tgz a un directorio temporal, introducirse en él y ejecutar:

```
make && make test && sudo make install
```

make test no es estrictamente necesario, pero es útil como prueba de que todo funciona como debería. Una vez termine, esto debería indicar algo relativo a que el símbolo "_start" no está definido (se correspondería con la función main de un programa C), indicando que todo ha ido bien:

```
ld -lga
```

Si fuera mal, tendríamos que mirar en /etc/ld.so.conf, y comprobar que la ruta bajo la que se ha instalado la biblioteca está allí.

Ahora podemos ir a instalacion-gamera/GAmera y compilarlo:

```
make
```

Ya sólo queda comprobar que GAmera en conjunto va bien. Para ello, seguiremos los siguientes pasos:

1. Copiar instalacion-gamera/GAmera/gamera.conf al directorio del ejemplo que se quiera probar. Por ejemplo:

   ```
   cp instalacion-gamera/GAmera/gamera.conf instalacion-gamera/test_processes/
       loanStructured
   ```

2. Modificar el gamera.conf con las rutas correctas al .bpel y .bpts, y poniendo ceros al lado de ANALIZADOR y BPELOUT, para que se generen automáticamente:

```
ANALIZADOR 0 analisis.txt
BPELORIGINAL 1 loanStructuredNonRPC.bpel
BPELOUT 0 loanStructuredNonRPC.bpel.out
TESTSUITE 1 ServicioPrestamosTest.bpts
```

3. Ejecutar GAmera. Desde el ejemplo antes indicado, sería:

```
../../ gamera
```

8.2. Instalación de Koura

Ya que tenemos instalada GAmera, que es la base de nuestra aplicación, vamos a instalar Koura.

8.2.1. Mono .Net

Ya que Koura está implementado en C♯, necesitaremos instalar los siguientes paquetes, empleamos la versión 2 de Mono):

1. mono-2.0-runtime
2. mono-runtime
3. mono-common
4. libmono2.0-cil
5. libmono-data2.0-cil
6. libmono-sharpzip2.84-cil

Para instalarlos bien lo haremos por consola o acudiremos al gestor de paquetes de nuestro sistema operativo Linux.

8.2.2. Perl

Aunque se incluyen en los ficheros de instalación de Koura los scripts de Perl que analizan las diferencias entre el fichero mutante y el fichero original, necesitaremos instalar Perl para que éstos puedan ejecutarse.

Los paquetes necesarios son los siguientes, aunque en este caso la versión de Perl no va a influir en la ejecución de Koura, podemos actualizarlo sin temor a que haya errores o incongruencias.

1. perl (versión 5.10.0)
2. perl-base (versión 5.10.0)

8.2.3. Librerías Gtk

Estas librerías son importantes ya que nos van a permitir la visualización de Koura, si no se tuviese la versión indicada de los paquetes que listaremos a continuación, no funcionaría correctamente Koura. La versión de las librerías Gtk tiene que ser como mínimo la que se indica en cada paquete:

1. checkbox-gtk 0.7.2
2. libmono-addins-gui0.2-cil
3. libgtk2.0-0
4. libgtk2.0-common
5. libgtk2.0-bin
6. libpango1.0-0 1.24.1
7. libpango1.0-common 1.24.1

8.2.4. Ubicación de los ficheros

Para ejecutar Koura, necesitaremos ubicar los ficheros en una carpeta predeterminada. Como ya hemos comentado, para que haya una mayor organización en los resultados, Koura se almacenará bajo la carpeta de GAmera ya que en GAmera estarán ubicados cada uno de los directorios de los casos de pruebas. En caso de que esta estructura no fuera posible o haya sido cambiada, Koura debe ser colocado en el directorio superior donde estén los casos de prueba “Koura/DirectoriosDeCasosDePrueba”. En nuestro caso lo tenemos situado en GAmera ya que seguimos la estructura de “GAmera/DirectoriosDeCasosDePrueba”.

Los ficheros que necesita Koura para su ejecución y que tienen que ubicarse en el directorio que se ha comentado antes son:

1. Ejecutable Koura
2. Fichero xmldiff.pl
3. Fichero xmlpp.pl
4. Directorio “img”

Bibliografía

[1] Estero Botaro, A., Palomo Lozano, F. y Medina Bulo, I. *Artículo - Operadores de mutación para WS-BPEL 2.0*,PRIS 2008.

[2] Domínguez Jiménez, J.J., Estero Botaro, A. y Medina Bulo, I. *Artículo - Generación de mutantes con algoritmos genéticos*,PRIS 2008.

[3] Medina Bulo, I., Domínguez Jiménez, J.J. y Estero Botaro, A. *Artículo - Un sistema para la generación automática de mutantes de composiciones WS-BPEL*, JSWEB 2009.

[4] Ceballos, Fco. Javier: El lenguaje de programación C. Ra-Ma 2002.

[5] `http://es.wikipedia.org`

[6] `http://cs.gmu.edu/~offutt/mujava/`

[7] `http://muclipse.sourceforge.net/`

[8] `http://agile.csc.ncsu.edu/SEMaterials/tutorials/muclipse/`

[9] `http://www.mono-hispano.org/wiki/Tutorial_de_XML_con_C_Sharp`

[10] `http://www.mono-project.com/Responsive_Applications`

[11] `http://www.javielinux.com/programacion_mono.php`

[12] `http://ftp.novell.com/pub/mono/sources-stable/`

[13] `http://tirania.wordpress.com/2008/11/21/instalar-mono-201-monodevelop-2-alpha2-en-debian-n_n/`

[14] `http://www.mono-hispano.org/wiki`

[15] `http://www.zetcode.com/tutorials/monowinformstutorial/`

[16] `http://go-mono.org/docs/`

[17] `http://mono-project.com/Guide:_Porting_Winforms_Applications`

[18] `http://www.iberprensa.com/todolinux/articulos/TL94-mono.pdf`

[19] http://www.go-mono.com/docs/index.aspx

[20] http://go-mono.com/forums/#nabble-to23991021

[21] http://msdn.microsoft.com/es-es/library/

[22] http://www.bravegnu.org/gtktext/

[23] http://www.soe.ucsc.edu/classes/cmps203/Spring06/Project/JavaCode/java-charlie/mono/mcs-0.23/tools/SqlSharp/gui/gtk-sharp/SqlEditorSharp.cs

[24] http://www.pygtk.org/pygtk2tutorial-es/sec-TextTagsAndTextTagTables.html

[25] http://mono.pointbeing.net/1.1/handlers/monodoc.ashx?link=T%3aGtk.TextIter%2f*

[26] http://developer.android.com/reference/android/widget/TextView.html

[27] http://es.wikibooks.org/wiki/C_sharp_NET/Capítulo_19

[28] http://java.sun.com/products/jlf/

[29] http://neptuno.uca.es/redmine/wiki/sources-fm/Instalacion_de_GAmera

Capítulo 9 Anexos

9.1. Anexo I

9.1.1. Perl - xmldiff

```
#!/usr/bin/perl
#
#   Copyright (c) 2002, DecisionSoft Limited All rights reserved.
#   Please see:
#    http://software.decisionsoft.com/licence.html
#   for more information.
#


#
# xmldiff: xmldiff program - uses xmlpp
#

#Change this if xmlpp is not in your current path
#for example: $XMLPP = "./xmlpp";
$XMLPP = "./xmlpp.pl";

# older versions of less don't support -R, consider -r instead
my $pagerCmd = ' | less -R ';

use Getopt::Std;

getopts('tsncupChHXSi');

if ($opt_h || @ARGV != 2) {
  usage();
}

my $diffOpts;

if ($opt_n) {
  $pagerCmd = '';
}

if ($opt_u + $opt_c + $opt_s + $opt_p + $opt_C > 1) {
  print STDERR "Error: Only one mode may be specified\n";
  usage();
}

```

```
39
40  # Set diff options
41  if ($opt_u) {
42    $diffOpts .= '-u';
43  }
44  if ($opt_c) {
45    $diffOpts .= '-c';
46  }
47
48  if (!$opt_s) {
49    $diffOpts = "--changed-group-format='%<%>' ";
50    $diffOpts .= " --new-group-format='%>' ";
51    $diffOpts .= "--old-group-format='%<' ";
52    $diffOpts .= "--new-line-format='[1m[33m+ %l\n[m' ";
53    $diffOpts .= "--old-line-format='[1m[31m- %l\n[m' ";
54    $diffOpts .= "--unchanged-line-format='[1m[30m %l[m\n' ";
55  }
56  if ($opt_C) {
57    $diffOpts = "--changed-group-format='\n<<<<<<<<<<<<<<<\n
          %<===============\n%>>>>>>>>>>>>>>>\n\n' ";
58    $diffOpts .= "--new-line-format='+ %l\n' ";
59    $diffOpts .= "--old-line-format='- %l\n' ";
60    $diffOpts .= "--unchanged-line-format=' %l\n' ";
61  }
62  if($opt_H){
63    $diffOpts = "--changed-group-format='%<%>' ";
64    $diffOpts .= " --new-group-format='%>' ";
65    $diffOpts .= "--old-group-format='%<' ";
66    $diffOpts .= "--new-line-format='<font color=\"green\">+ %l</font>\n' ";
67    $diffOpts .= "--old-line-format='<font color=\"red\">- %l</font>\n' ";
68    $diffOpts .= "--unchanged-line-format='<font color=\"gray\"> %l</font>\n'
          ";
69  }
70  if($opt_X){
71    $diffOpts = "--changed-group-format='%<%>' ";
72    $diffOpts .= " --new-group-format='%>' ";
73    $diffOpts .= "--old-group-format='%<' ";
74    $diffOpts .= "--new-line-format='#r %l\n' ";
75    $diffOpts .= "--old-line-format='#v %l\n' ";
76    $diffOpts .= "--unchanged-line-format=' %l\n' ";
77  }
78
79  # Set up xmlpp options
80  my $prettyOpts = $opt_t ? "-t " : "";
81  $prettyOpts .= $opt_S ? "-S " : "";
82  $prettyOpts .= $opt_H ? "-H " : "";
83  $prettyOpts .= $opt_X ? "-X " : "";
84  $prettyOpts .= "-s -e ";
85
86  $file1 = "xmlppTEMP1.$$";
87  $file2 = "xmlppTEMP2.$$";
```

```
88  system("$XMLPP $prettyOpts '$ARGV[0]' > $file1");
89  system("$XMLPP $prettyOpts '$ARGV[1]' > $file2");
90
91  if($opt_H) {
92
93
94    print <<EOF;
95  <HTML>
96    <HEAD>
97      <TITLE>XML diff</TITLE>
98    </HEAD>
99    <BODY bgcolor="#FFFFFF" >
100
101     <PRE>
102 EOF
103
104   system("diff -bB $diffOpts $file1 $file2");
105
106 print <<EOF;
107     </PRE>
108   </BODY>
109 </HTML>
110 EOF
111
112 }
113 else {
114
115   system("diff -bB $diffOpts $file1 $file2 $pagerCmd");
116 }
117
118
119 unlink($file1,$file2);
120
121 exit($? >> 8);
122
123 sub usage {
124   print STDERR <<EOF;
125 usage: $0 [ mode ] [ options ] oldfile.xml newfile.xml
126
127 Warning: The exit code from xmldiff is only meaningful if run with the
        -n
128 option.
129
130 mode must be one of:
131   -p coloured unified diff [default]
132   -c context diff
133   -u unified diff
134   -s standard diff output
135   -C vaguely CVS like unified diff
136
137 options:
```

```
  -H HTML output
  -X XML output
  -t split attributes - good for spotting changes in attributes
  -n don't pipe output through less
  -S schema hack mode - good for diffing schemas
  -i ignore element and attribute contents

EOF
  exit 1;
}
```

Listado 9.1: xmldiff.pl

9.1.2. Perl - xmlpp.pl

```
#!/usr/bin/perl -w

#
#  Copyright (c) 2002, DecisionSoft Limited All rights reserved.
#  Please see:
#  http://software.decisionsoft.com/licence.html
#  for more information.
#

# $Revision: 1.32 $
#
# xmlpp: XML pretty printing
#

# For custom attribute sorting create an attributeOrdering.txt file that
# lists each attributes separated by a newline in the order you would like
# them to be sorted separated by a newline. Then use the -s option.

use FileHandle;
use Fcntl;
use Getopt::Std;

use vars qw($opt_h $opt_H $opt_X $opt_s $opt_z $opt_t $opt_e $opt_S $opt_c $opt_n);

my $indent=0;
my $textContent='';
my $lastTag=undef;
my $output;
my $inAnnotation = 0;


if (!getopts('nzhHsteXSc') or $opt_h) {
```

```
33      usage();
34  }
35
36  if  ($opt_s){
37
38  # expect to find attributeOrdering.txt file in same directory
39  # as xmlpp is being run from
40
41    my $scriptDir = $0;
42    if  ($scriptDir =~ m#/#){
43      $scriptDir =~ s#/[^/]+$##;
44    }
45    else{
46      $scriptDir ="."; 
47    }
48
49    # get attribute ordering from external file
50    if  (open(SORTLIST, "<$scriptDir/attributeOrdering.txt")) {
51      @sortlist = <SORTLIST>;
52      chomp @sortlist;
53      close (SORTLIST);
54      @specialSort = grep(/^\w+/, @sortlist);
55    }
56    else {
57     print STDERR "Could not open $scriptDir/attributeOrdering.txt: $!\
          nWARNING attribute sorting will only be alphabetic\n\n";
58    }
59  }
60
61
62  # set line separator to ">" speeding up parsing of XML files
63  # with no line breaks
64
65  $/ = ">";
66
67
68  my $sortAttributes = $opt_s;
69  my $newLineComments = $opt_c;
70  my $splitAttributes = $opt_t;
71  my $schemaHackMode = $opt_S;
72  my $normaliseWhiteSpace = $opt_n;
73
74  my $filename = $ARGV[0];
75  if  ($opt_z && (!$filename or $filename eq '-')) {
76      print STDERR "Error: I can't edit STDIN in place.\n";
77      usage();
78  }
79
80  if  (!$opt_z && scalar(@ARGV) > 1) {
81      print STDERR "Warning: Multiple files specified without -z option\n";
82  }
```

```
83
84  my $fh;
85
86  my $stdin;
87
88  if (!$filename or $filename eq '-') {
89      $fh=*STDIN;
90      $stdin=1;
91  } else {
92      $fh = open_next_file() or exit(1);
93      $stdin=0;
94  }
95
96  do {
97      $indent=0;
98      $textContent='';
99      $lastTag=undef;
100     $output = '';
101     my $re_name = "(?:[A-Za-z0-9_:][A-Za-z0-9_:.-]*)";
102     my $re_attr = "(?:'[^']*'|\"[^\"]*\")";
103     my $input;
104
105     while ($input .= <$fh>) {
106        while ($input) {
107           if ($input =~ s/^<($re_name)((?:\s+$re_name\s*=\s*$re_attr)*\s
               *)(\/?)>(.*)$/$4/s ) {
108              my %attr;
109              my ($name,$attr,$selfclose) = ($1,$2,$3);
110              while ($attr =~ m/($re_name)\s*=\s*($re_attr)/gs) {
111                 my ($name,$value) = ($1,$2);
112                 $value =~ s/^["']([^"']*)["']$/$1/s;
113                 $attr{$name} = $value;
114              }
115              if ($opt_e) {
116                 parseStart($name, 0, %attr);
117                 if ($selfclose) { parseEnd($name) }
118              } else {
119                 parseStart($name, $selfclose, %attr);
120              }
121           } elsif ($input =~ s/^<\/($re_name)\s*>(.*)$/$2/s) {
122              parseEnd($1);
123           } elsif ($input =~ s/^<!--(.*?)-->(.*)$/$2/s) {
124              parseComment($1);
125           } elsif ($input =~ s/^([^<]+)(.*)$/$2/s) {
126              parseDefault($1);
127           } elsif ($input =~ s/^(<\?[^>]*\?>)(.*)$/$2/s) {
128              parsePI("$1\n");
129           } elsif ($input =~ s/^(<\!DOCTYPE[^\[>]*(\[[^\]]*\])?[^>]*>)(.*)$/$3/s) {
130              parseDoctype("$1");
131           } else {
132              last;
```

```
133             }
134         }
135         if (eof($fh)) {
136             last;
137         }
138     }
139
140
141     if ($input) {
142         $input =~ m/([^\n]+)/gs;
143         print STDERR "WARNING: junk remaining on input: $1\n";
144     }
145     $fh->close();
146
147     if (!$opt_z) {
148         if (!$opt_H && $opt_X){
149             print "$output\n"
150         } else {
151             print html_escape($output)."\n"
152         }
153     } else {
154         if ($input) {
155             print STDERR "Not overwriting file\n";
156         } else {
157             open FOUT,"> $filename" or die "Cannot overwrite file: $!";
158             if (!$opt_H && !$opt_X){
159                 print FOUT "$output\n"
160             }else {
161                 print FOUT html_escape($output)."\n"
162             }
163             close FOUT
164         }
165     }
166 } while (
167     !$stdin && $opt_z && ($fh = open_next_file(\$filename))
168   );
169
170
171
172 sub parseStart {
173     my $s = shift;
174     my $selfclose = shift;
175     my %attr = @_;
176
177     $textContent =~ s/\s+$//;
178     printContent($textContent);
179
180     if ($inAnnotation) {
181         return;
182     }
183
```

```
184     if ($schemaHackMode and $s =~ m/(^|:)annotation$/) {
185         $inAnnotation = 1;
186         $textContent = '';
187         $lastTag = 1;
188         return;
189     }
190     if (length($output)) {
191         $output .= "\n";
192     }
193
194     $output .= "  " x $indent;
195     $output .= "<$s";
196     my @k = keys %attr;
197
198     if ($sortAttributes && (scalar(@k) > 1) ){
199
200       my @alphaSorted;
201       my @needSpecialSort;
202       my @final;
203       my $isSpecial;
204
205       # sort attributes alphabetically (default ordering)
206       @alphaSorted = sort @k;
207
208       # read through sorted list, if attribute doesn't have specified
209       # sort order, push it onto the end of the final array (this
              maintains
210       # alphabetic order). Else create a list that has attributes needing
211       # special ordering.
212       foreach $attribute (@alphaSorted){
213         $isSpecial = 0;
214         foreach $sortAttrib (@specialSort){
215           if ($attribute eq $sortAttrib){
216             push @needSpecialSort, $attribute;
217             $isSpecial = 1;
218           }
219         }
220         if (!$isSpecial){
221           push @final, $attribute;
222         }
223       }
224
225       # now read through the specialSort list backwards looking for
226       # any match in the needSpecialSort list. Unshift this onto the
227       # front of the final array to maintain proper order.
228       foreach my $attribute (reverse @specialSort){
229         foreach (@needSpecialSort){
230           if ($attribute eq $_){
231             unshift @final, $attribute;
232           }
233         }
```

```
234       }
235
236       @k = @final;
237     }
238
239     foreach my $attr (@k) {
240         #
241         # Remove (min|max)Occurs = 1 if schemaHackMode
242         #
243         if ($schemaHackMode and $attr =~ m/^(minOccurs|maxOccurs)$/ and $
              attr{$attr} eq "1") {
244             next;
245         }
246
247         if ($splitAttributes) {
248             $output .= "\n"." " x $indent." ";
249         }
250         if ($attr{$attr} =~ /'/){
251             $output .= " $attr=\"$attr{$attr}\"";
252         } else {
253             $output .= " $attr='$attr{$attr}'";
254         }
255     }
256     if ($splitAttributes and @k) {
257         $output .= "\n"." " x $indent;
258     }
259     if ($selfclose) {
260         $output .= " />";
261         $lastTag = 0;
262     } else {
263         $output .= ">";
264         $indent++;
265         $lastTag = 1;
266     }
267     $textContent = '';
268 }
269
270 sub parseEnd {
271     my $s = shift;
272
273     if ($inAnnotation) {
274         if ($s =~ m/(^|:)annotation$/) {
275             $inAnnotation = 0;
276         }
277         return;
278     }
279
280     if ($normaliseWhiteSpace) {
281         $textContent =~ s/^\s*(.*?)\s*$/$1/;
282     }
283     $indent--;
```

```
284     printContent($textContent);
285     if ($lastTag == 0) {
286         $output .= "\n";
287         $output .= "  " x $indent;
288     }
289     $output .= "</$s>";
290     $textContent = '';
291     $lastTag = 0;
292 }
293
294 sub parseDefault {
295     my $s = shift;
296     if ($inAnnotation) { return }
297     $textContent .= "$s";
298 }
299
300 sub parsePI {
301     my $s = shift;
302     $output .= "$s";
303 }
304
305 sub parseDoctype {
306     my $s = shift;
307     if ($s =~ /^([^\[]*\[)([^\]]*)(\].*)$/ms) {
308       $start = $1;
309       $DTD = $2;
310       $finish = $3;
311       $DTD =~ s/\</\n \</msg;
312       $output .= "$start$DTD\n$finish\n";
313     } else {
314       $output .= "$s";
315     }
316 }
317
318 sub parseComment {
319     my $s = shift;
320     if ($inAnnotation) { return }
321     printContent($textContent,1);
322     if ($s =~ /([^\<]*)(<.*>)(.*)/ms) {
323       $start = $1;
324       $xml = $2;
325       $finish = $3;
326       $xml =~ s/\</\n\</msg;
327       $xml =~ s/(\n\s*\n?)+/\n/msg;
328       $xml =~ s/^\s*//msg;
329       $xml =~ s/\s*$//msg;
330       $s = "$start\n$xml\n$finish";
331     }
332     $s =~ s/\n\s*$/\n /msg;
333     if ($newLineComments) {
334         $output .= "\n<!--$s-->\n";
```

```
335     } else {
336         $output .= "<!--$s-->";
337     }
338     $textContent='';
339 }
340
341 sub printContent {
342     my $s = shift;
343     my $printLF = shift;
344     my ($LF,$ret) = ("","");
345
346     if ($s =~ m/\n\s*$/) {
347         $LF = "\n";
348     }
349     if ($s =~ m/^[\s\n]*$/) {
350         $ret = undef;
351     } else {
352         $output .= "$s";
353         $ret = 1;
354     }
355     if ($printLF) {
356         $output .= $LF;
357     }
358 }
359
360
361 sub html_escape {
362     my $s = shift;
363     $s =~ s/&/&amp;/gsm;
364     $s =~ s/</&lt;/gsm;
365      $s =~ s/>/&gt;/gsm;
366     return $s;
367 }
368
369 sub open_next_file {
370     my $filename = shift;
371     $$filename = shift @ARGV;
372     while ($$filename and ! -f $$filename) {
373         print STDERR "WARNING: Could not find file: $$filename\n";
374         $$filename = shift @ARGV;
375     }
376     if (!$$filename) {
377         return undef;
378     }
379     my $fh = new FileHandle;
380     $fh->open("< $$filename") or die "Can't open $$filename: $!";
381     return $fh;
382 }
383
384 sub usage {
385     print STDERR <<EOF;
```

```
386 usage: $0 [ options ] [ file .xml ... ]
387
388 options:
389   -h display this help message
390   -H escape characters (useful for further processing)
391   -X Xml
392   -t split attributes, one per line (useful for diff)
393   -s sort attributes (useful for diff)
394   -z in place edit (zap)
395   -e expand self closing tags (useful for diff)
396   -S schema hack mode (used by xmldiff)
397   -c place comments on new line.
398   -n normalise whitespace (remove leading and trailing whitespace from nodes
399       with text content.
400
401 EOF
402     exit 1;
403 }
```

Listado 9.2: xmlpp.pl

9.2. Anexo II

9.2.1. Introducción al entorno MonoDevelop

En este anexo vamos a introducir al lector un poco en el entorno de MonoDevelop. Para su instalación iremos al gestor de paquetes correspondiente a nuestra distribución de Linux y buscaremos el paquete "monodevelop". Lo instalaremos y una vez instalado nos aparecerá entre nuestras aplicaciones la herramienta "MonoDevelop".

9.2.2. Bienvenido

La página de inicio de Monodevelop es bastante intuitiva y facilita mucho su manejo. Se nos presenta cuatro áreas de tareas:

1. Acciones típicas
2. Soluciones recientes
3. Enlaces de ayuda
4. Enlaces sobre desarrollo

En la primera sección están las opciones de abrir una solución (nombre que reciben los proyectos elaborados en MonoDevelop) o fichero. O bien te permite comenzar un nuevo proyecto o solución.

En la segunda sección aparecerá un listado (dependiendo del uso que le demos a esta herramienta) con las soluciones más recientes para tenerlas más a mano a la hora de entrar en MonoDevelop.

En la tercera sección son los enlaces de ayuda que ofrece el programa, una de ellas es para la página oficial del Proyecto Mono y la otra es para la página oficial de MonoDevelop.

En la cuarta sección son enlaces para la ayuda del desarrollo de una aplicación en MonoDevelop. Se presentan páginas de librerías de Mono, enlaces sobre artículos .NET...

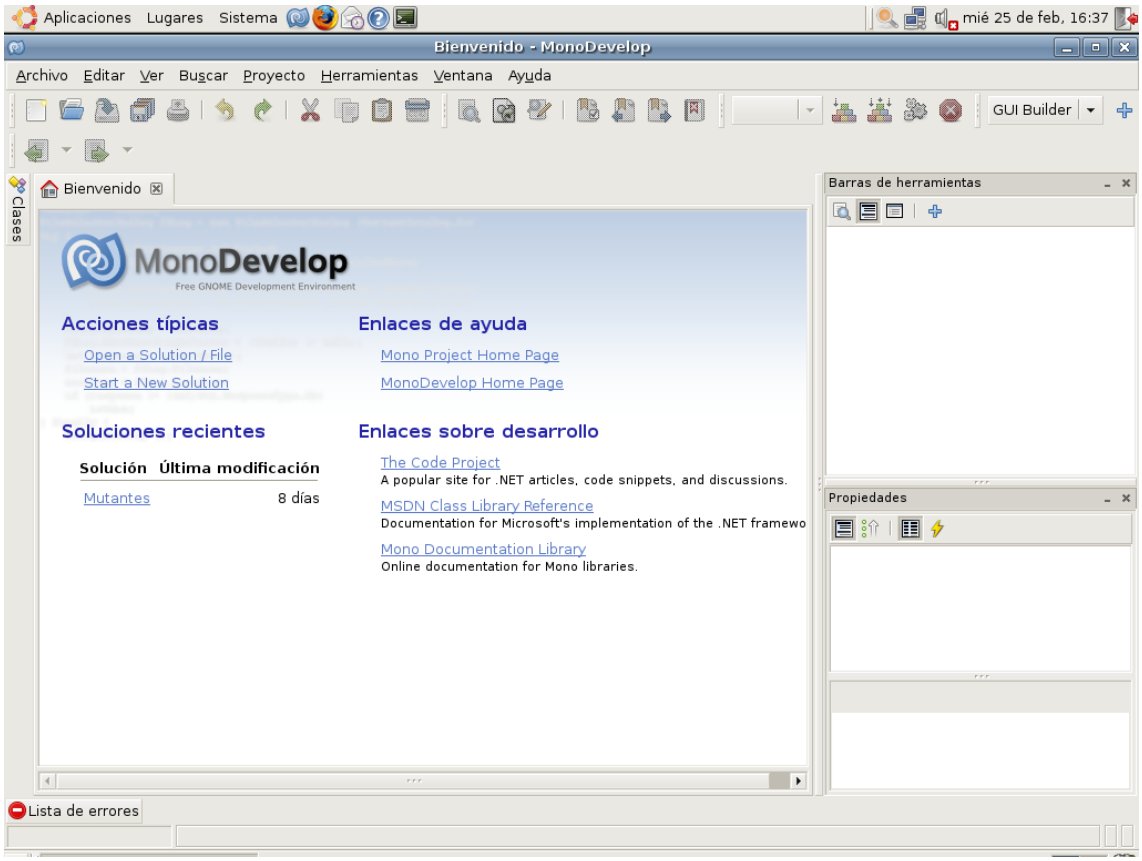

Figura 9.1: Captura MonoDevelop - Bienvenida

9.2.3. Entorno de desarrollo - Código fuente

Una vez que tenemos un proyecto creado, podemos observar que hay en la parte inferior dos modos de trabajar con la solución. Una parte es la del "Código fuente", y la otra es la de "Diseñador".

En esta sección podemos implementar las funcionalidades de nuestros componentes y modificar su comportamiento según las acciones que pueda realizar el usuario con dicho componente. Aparece en la parte superior un listado con las diferentes funcionalidades que hemos implementado para acceder a ellas de forma más directa.

Mientras que estemos implementando, se nos va a ofrecer un desplegable que es una ayuda con las diferentes opciones que se nos presentan según el componente o función que estemos implementando. Del mismo modo junto a este listado, según el elemento que esté seleccionado en esta lista, aparecerá el significado y su modo de empleo.

Para que aparezca en la parte de "Código fuente" la nueva función que queremos implementar (a partir de un componente dado), tendríamos que hablar desde la parte de "Diseñador" en la que seleccionando el componente a implementar, se nos aparece en "Propiedades", en la pestaña "Señales" las diferentes operaciones que se pueden realizar con el componente seleccionado. Haciendo doble clik sobre la función se nos abrirá la parte de "Código fuente" con la nueva función creada lista para que empecemos a implementar.

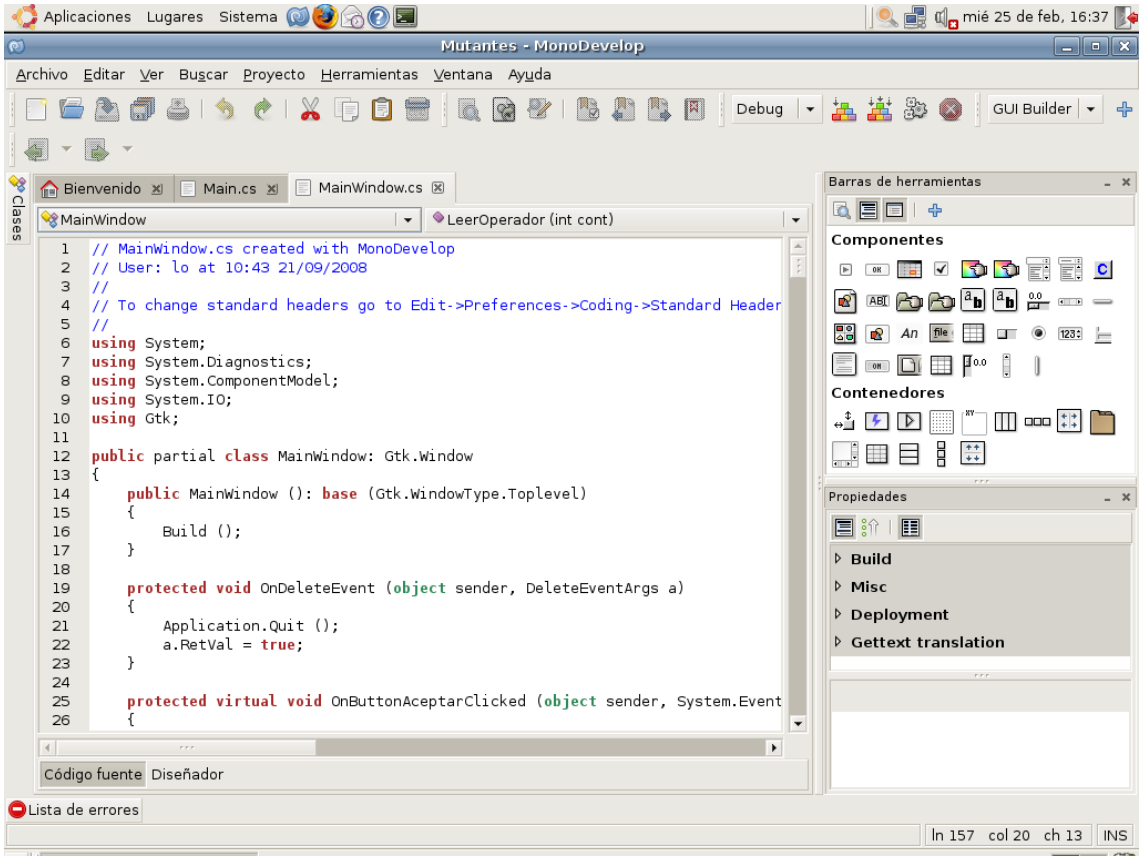

Figura 9.2: Captura MonoDevelop - Código fuente

9.2.4. Entorno de desarrollo - Diseñador

En esta sección tenemos que ir añadiendo los componentes que necesitemos en el entorno que se nos presenta. Los diferentes componentes que se nos ofertan son los que aparecen en el lateral derecho "Barra de herramientas". Seleccionaremos el componente y luego lo arrastraremos hasta la zona donde queremos que se sitúe. Para añadirle propiedades hacemos clik en el componente y en "Propiedades" aparece un listado con todos los valores que se le pueden aplicar. Esta modificación de valores se aplican directamente sobre los componentes. También podemos acceder a las definiciones de los valores en el fichero "Main.cs", pero por comodidad se utiliza el listado de "Propiedades".

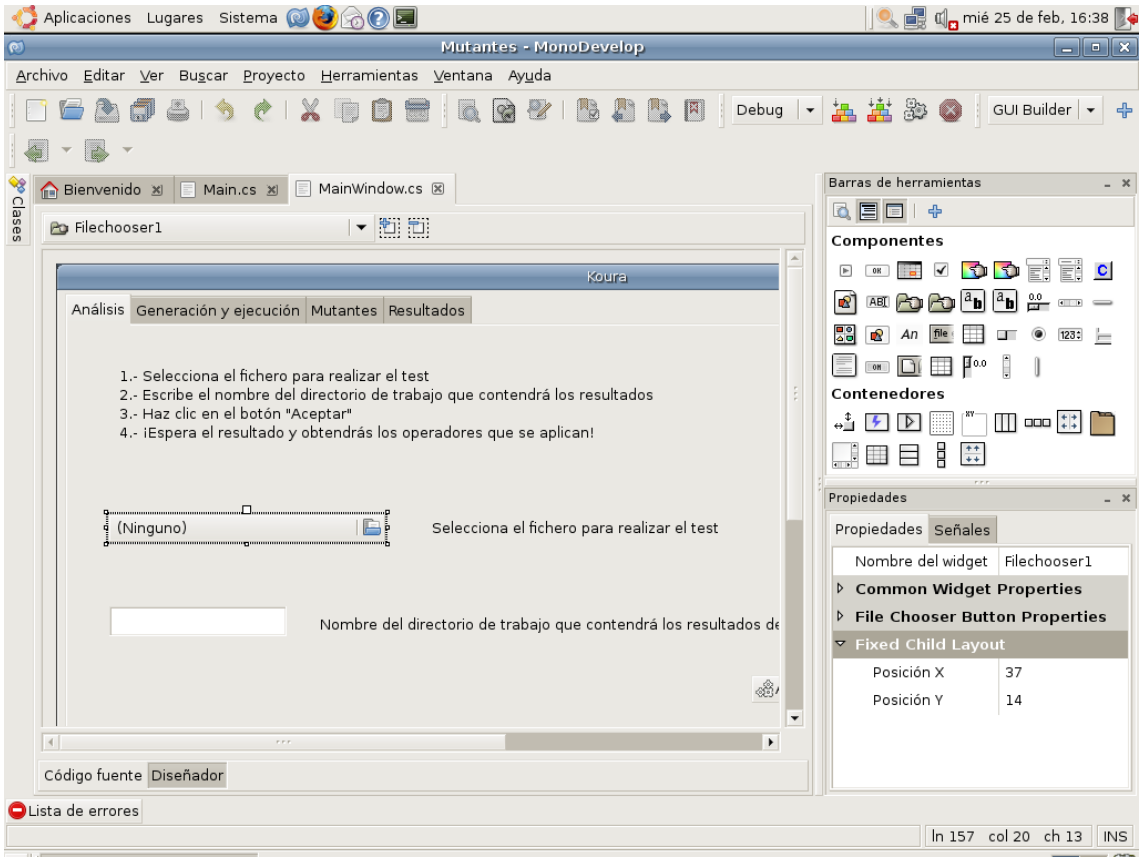

Figura 9.3: Captura MonoDevelop - Diseñador

Para compilar la solución tenemos dos opciones, haciendo clik en el botón de compilar que aparece el la barra de menú superior o bien en Proyecto - Compilar. Y para echar a andar el proyecto y ejecutarlo igualmente haciendo clik en el botón de ejecutar de la barra de menú superior o bien en Proyecto - Ejecutar.

Si hay algún error durante la ejecución o durante la compilación aparece en el desplegable de "Lista de errores".